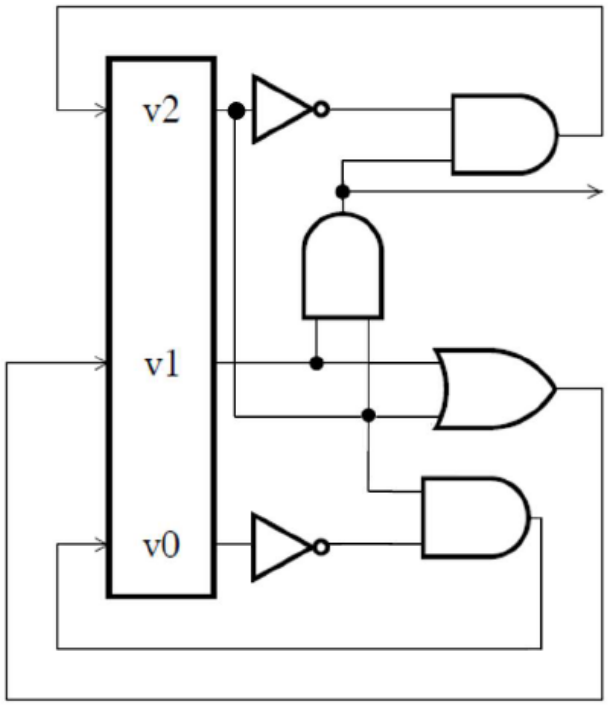


The initial value of the state variable v_0 of the circuit is $false$. The initial values of $v_1$and v_2 are unknown.

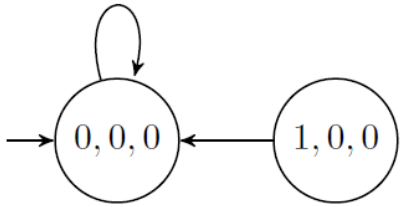
Task 1a. [5 Points]

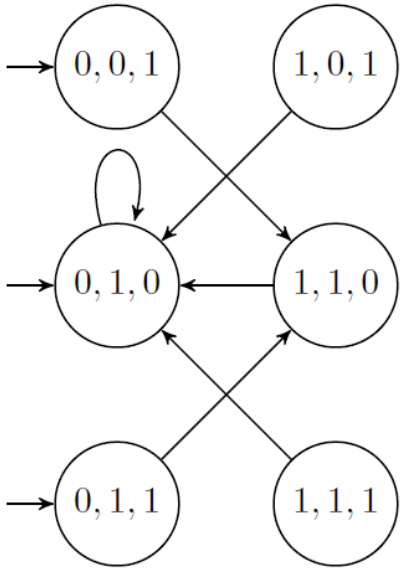
- State the formula S_0 that represents the set of initial states and the formula R that represents the transition relation of C.

Task 1b. [5 Points]

- Draw the Kripke Structure $M = (\mathrm{S}, \mathrm{S}_0, \mathrm{R}, \mathrm{AP}, \mathrm{L})$ that represents C.

v0, v1, v2





taken from an anonymous student

SAT-Based Model Checking

Chapter 10

Outline

- Bounded Model Checking
- Verifying Reachability Properties with k-induction
- Model Checking with Inductive Invariants
- Model Checking with Craig Interpolants
- Property-Directed Reachability

Overview

- SAT solvers can solve propositional formulas. (See Chapter 9.)
- SAT solvers have become very fast

Techniques

- Bounded Model Checking: Is there a trace of length k that violates the property?
- k-induction: Can we prove the property inductively for *any* trace?
- Create an inductive invariant that is stronger than the property
  - Craig Interpolants
  - Property-Directed Reachability

In this chapter, we will only consider LTL formulas

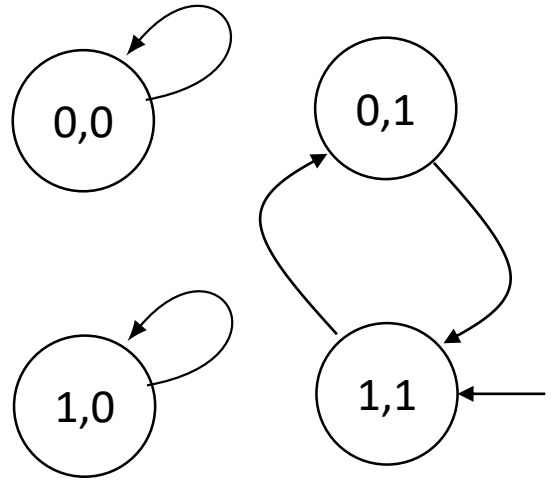
Preliminaries

Postimage

postimg(S) = states reachable from S in one step

postimg({ (0,1) }) =

postimg({ (0,0), (1,1)}) =



Kripke structure

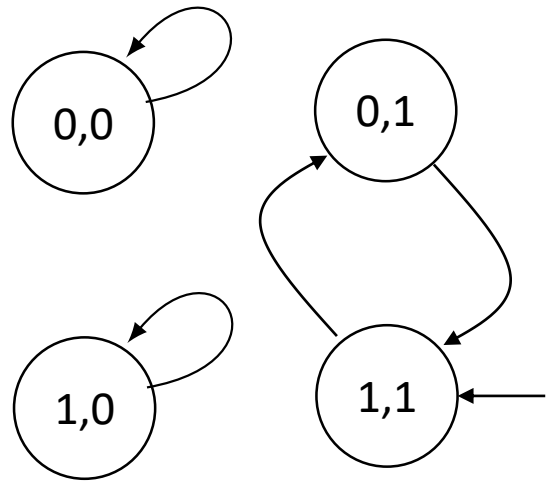
Paths

Initial states: $\mathcal{S}_0(x, y) = (x = 1 \wedge y = 1)$

Transitions: $\mathcal{R}\ (x, y, x', y') = (x' = (x + y)\ mod\ 2) \wedge (y' = y)$

$Path_1(S(V)) = S \wedge R(V, V') =.$

$\exists V. \neg x \wedge y \wedge (x' = (x + y)\ mod\ 2) \wedge (y' = y) =$



Kripke structure

Paths

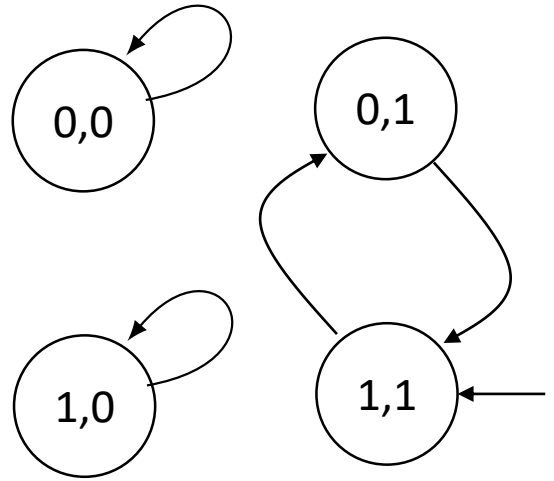
Initial states: $\mathcal{S}_0(x, y) = (x = 1 \wedge y = 1)$

Transitions: $\mathcal{R}\ (x, y, x', y') = (x' = (x + y)\ mod\ 2) \wedge (y' = y)$

$Path_1(S(V)) = S \wedge R(V, V') =.$

$x \wedge y \wedge (x' = (x + y)\ mod\ 2) \wedge (y' = y) =.$

$x \wedge y \wedge \neg x' \wedge y'.$



Kripke structure

Bounded Model Checking

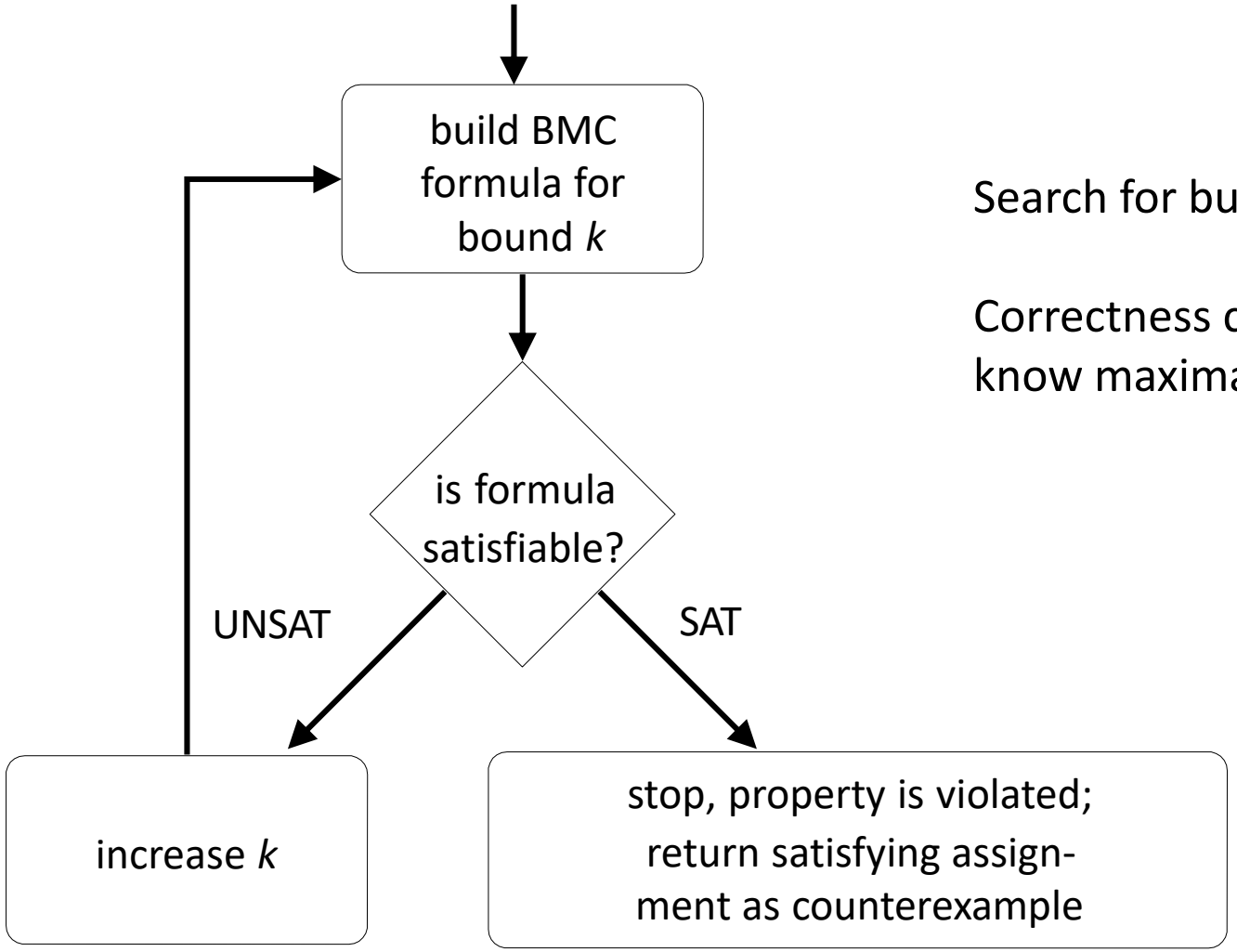
Computer Aided Verification Award 2018

Bounded Model Checking has revolutionized the way model checking is used and perceived. It has increased the capabilities of model checkers by orders of magnitude, turning them into a standard tool for hardware verification and a very important component of the toolkit available for software verification…

(1999)

Flavio Lerda, Daniel Kroening, Armin Biere, Alessandro Cimatti,
Not pictured: Edmund M. Clarke, Yunshan Zhu

Bounded Model Checking



Search for bugs within k steps.

Correctness can only be proven if we know maximal value for *k*

Reachability Properties

Property

p always holds iff we cannot reach a state with $\neg p$

Kripke Structure

$M = (S, S_0, R, AP, L)$ – represented symbolically (See Chapter 3)

State variables $\mathrm{V} = \{v_1, \ldots, v_n\}$

Reachability – Paths

$path_0(s_0) =$

$path_1(s_0, s_1) =$

$path_2(s_0, \ldots, s_2) =$

$path_k(s_0, \ldots, s_k) =$

Reachability – Paths

$$path_0(s_0) = S_0(s_0)$$
$$path_1(s_0, s_1) = S_0(s_0) \wedge R(s_0, s_1)$$
$$path_2(s_0, \ldots, s_2) = S_0(s_0) \wedge R(s_0, s_1) \wedge R(s_1, s_2)$$
$$path_3(s_0, \ldots, s_3) = S_0(s_0) \wedge R(s_0, s_1) \wedge R(s_1, s_2) \wedge R(s_1, s_3)$$

$$path_k(s_0, \ldots, s_k) = S_0(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1})$$

Reachability – Building the Formula

Reachability – Building the Formula

$$path_k(s_0, \ldots, s_k) = S_0(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1})$$

Path starts in initial state

Exists transition from s_i to s_{i+1}

System is **incorrect** within k steps if

$$path_k(s_0, \ldots, s_k) \wedge \bigvee_{i=0}^{k} \neg p(s_i)$$

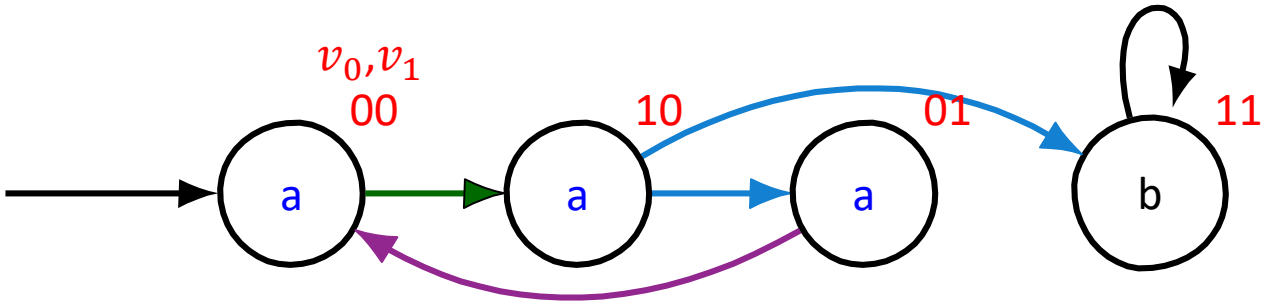
There is a path to s_k

One of the state violates p

Reachability – Correctness

Theorem 10.1. $path_{k(s_0,\ldots,s_k)} \wedge \bigvee_{i=0}^{k} \neg p(s_i)$ is satisfiable iff there is a counterexample to $\mathrm{AG}\, p$ of length $\leq k$.

Example



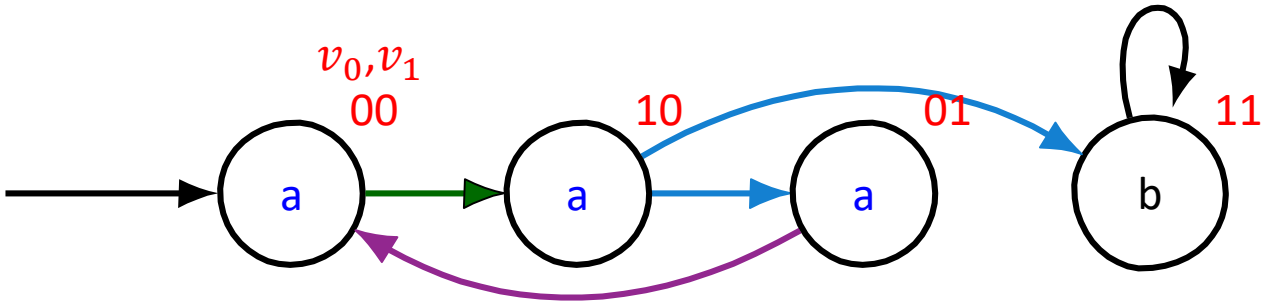
AG a

$S_0(v_0, v_1) =.$

$R(v_0, v_1, v_0', v_1')$ =

$a(v_0, v_1) =.$

Example



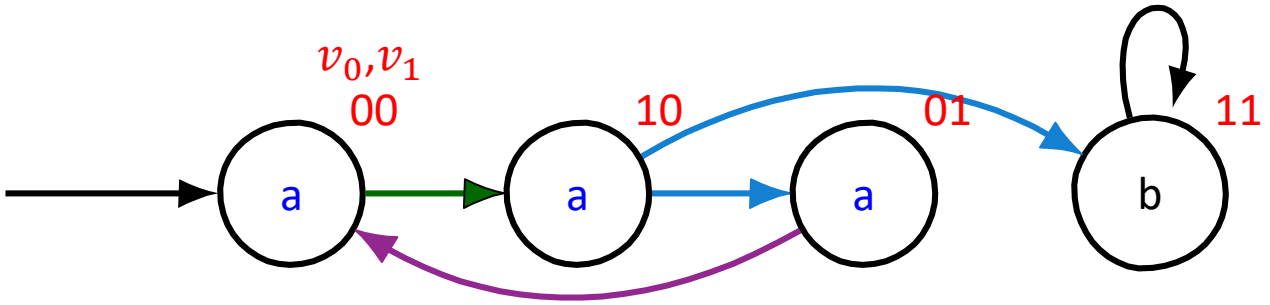
AG a

$$S_0(v_0, v_1) = \neg v_0 \wedge \neg v_1$$

$$R(v_0, v_1, v_0', v_1') = \neg v_0 \wedge \neg v_1 \wedge v_0' \wedge \neg v_1'$$
$$\vee\ v_0 \wedge \neg v_1 \wedge v_1'$$
$$\vee\ \neg v_0 \wedge v_1 \wedge \neg v_0' \wedge \neg v_1'$$
$$\vee\ v_0 \wedge v_1 \wedge v_0' \wedge v_1'$$

$$a(v_0, v_1) = \neg v_0 \vee \neg v_1$$

Example



$$S_0(v_0, v_1) = \neg v_0 \wedge \neg v_1$$

$$R(v_0, v_1, v_0', v_1') = \neg v_0 \wedge \neg v_1 \wedge v_0' \wedge \neg v_1' \vee v_0 \wedge \neg v_1 \wedge v_1' \vee \neg v_0 \wedge v_1 \wedge \neg v_0' \wedge \neg v_1' \vee v_0 \wedge v_1 \wedge v_0' \wedge v_1'$$

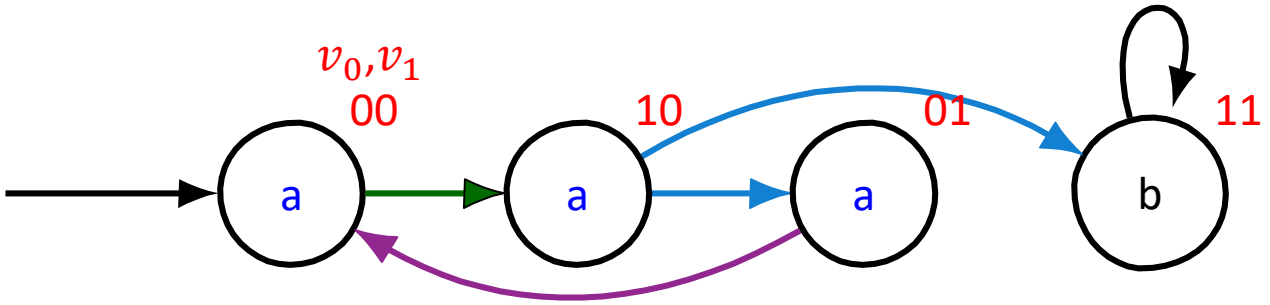
$$a(v_0, v_1) = \neg v_0 \vee \neg v_1$$

$$path_1(s_0, s_1, s_2) = S_0(s_0) \wedge R(s_0, s_1) \wedge R(s_1, s_2)$$

$$= \neg v_{0,0} \wedge \neg v_{1,0} \wedge \begin{pmatrix} \neg v_{00} \wedge \neg v_{10} \wedge v_{01} \wedge \neg v_{11} \\ \vee\ v_{00} \wedge \neg v_{10} \wedge v_{11} \\ \vee\ \neg v_{00} \wedge v_{10} \wedge \neg v_{01} \wedge \neg v_{11} \\ \vee\ v_{00} \wedge v_{10} \wedge v_{01} \wedge v_{11} \end{pmatrix}$$

$$\bigvee_{i=0}^{k} \neg a(s_i) = \neg(\neg v_{00} \vee \neg v_{10}) \vee \neg(\neg v_{01} \vee \neg v_{11})$$

Example



$$S_0(v_0, v_1) = \neg v_0 \wedge \neg v_1$$

$$R(v_0, v_1, v_0', v_1') = \neg v_0 \wedge \neg v_1 \wedge v_0' \wedge \neg v_1' \vee v_0 \wedge \neg v_1 \wedge v_1' \vee \neg v_0 \wedge v_1 \wedge \neg v_0' \wedge \neg v_1' \vee v_0 \wedge v_1 \wedge v_0' \wedge v_1'$$

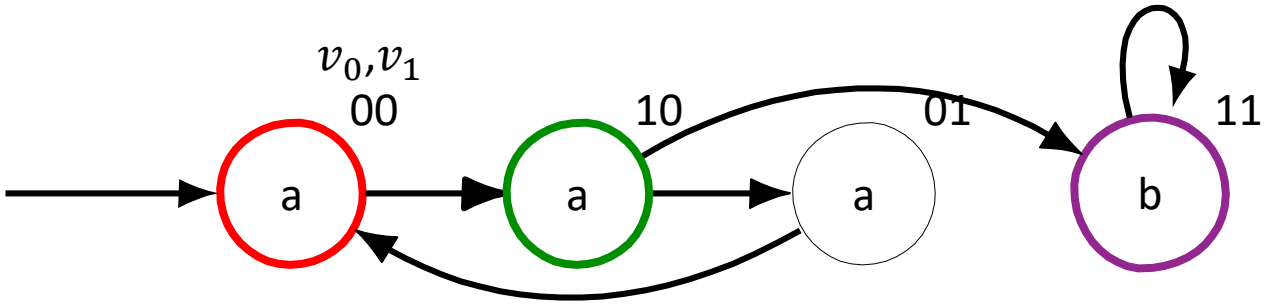
$$a(v_0, v_1) = \neg v_0 \vee \neg v_1$$

$$path_2(s_0, s_1, s_2) = S_0(s_0) \wedge R(s_0, s_1) \wedge R(s_1, s_2)$$

$$= \neg v_{0,0} \wedge \neg v_{1,0} \wedge \begin{pmatrix} \neg v_{00} \wedge \neg v_{10} \wedge v_{01} \wedge \neg v_{11} \\ \vee\ v_{00} \wedge \neg v_{10} \wedge v_{11} \\ \vee\ \neg v_{00} \wedge v_{10} \wedge \neg v_{01} \wedge \neg v_{11} \\ \vee\ v_{00} \wedge v_{10} \wedge v_{01} \wedge v_{11} \end{pmatrix} \wedge \begin{pmatrix} \neg v_{01} \wedge \neg v_{11} \wedge v_{02} \wedge \neg v_{12} \\ \vee\ v_{01} \wedge \neg v_{11} \wedge v_{12} \\ \vee\ \neg v_{01} \wedge v_{11} \wedge \neg v_{02} \wedge \neg v_{12} \\ \vee\ v_{01} \wedge v_{11} \wedge v_{02} \wedge v_{12} \end{pmatrix}$$

$$\bigvee_{i=0}^{k} \neg a(s_i) = \neg(\neg v_{00} \vee \neg v_{10}) \vee \neg(\neg v_{01} \vee \neg v_{11}) \vee \neg(\neg v_{02} \vee \neg v_{12})$$

Example



AG a

$$path_2(s_0, s_1, s_2) = S_0(s_0) \wedge R(s_0, s_1) \wedge R(s_1, s_2)$$
$$= \neg v_{0,0} \wedge \neg v_{1,0} \wedge$$
$$\left(\begin{array}{c} \neg v_{00} \wedge \neg v_{10} \wedge v_{01} \wedge \neg v_{11} \\ \vee\ v_{00} \wedge \neg v_{10} \wedge v_{11} \\ \vee\ \neg v_{00} \wedge v_{10} \wedge \neg v_{01} \wedge \neg v_{11} \\ \vee\ v_{00} \wedge v_{10} \wedge v_{01} \wedge v_{11} \end{array} \right) \wedge$$
$$\left(\begin{array}{c} \neg v_{01} \wedge \neg v_{11} \wedge v_{02} \wedge \neg v_{12} \\ \vee\ v_{01} \wedge \neg v_{11} \wedge v_{12} \\ \vee\ \neg v_{01} \wedge v_{11} \wedge \neg v_{02} \wedge \neg v_{12} \\ \vee\ v_{01} \wedge v_{11} \wedge v_{02} \wedge v_{12} \end{array} \right)$$

$$\bigvee_{i=0}^{k} \neg a(s_i) = \neg(\neg v_{00} \vee \neg v_{10}) \vee \neg(\neg v_{01} \vee \neg v_{11}) \vee \neg(\neg v_{02} \vee \neg v_{12})$$

Satisfying assignment

i	0	1	2
v_{0i}	0	1	1
v_{1i}	0	0	1

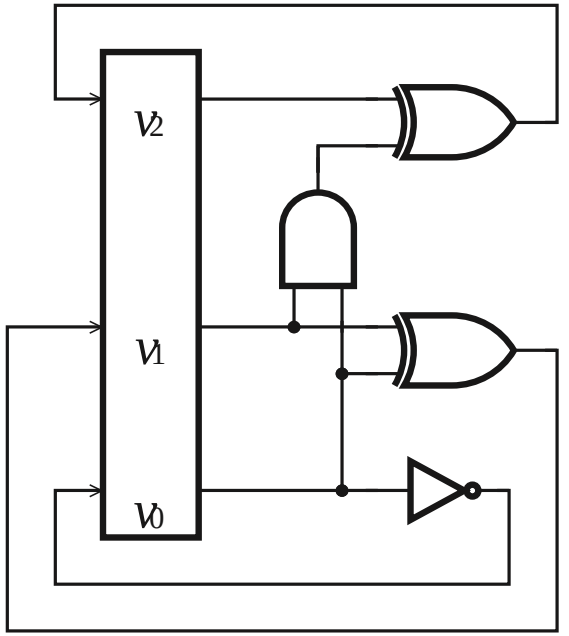
Try it with Z3!

```
(declare-const v00 Bool)
(declare-const v10 Bool)
(declare-const v01 Bool)
(declare-const v11 Bool)
(declare-const v02 Bool)
(declare-const v12 Bool)

(define-fun S0 ((v0 Bool) (v1 Bool)) Bool
  (and (not v0) (not v1)))

(define-fun R ((v0 Bool) (v1 Bool) (w0 Bool) (w1 Bool))
Bool
  (or
      (and (not v0) (not v1) w0 (not w1))
      (and v0 (not v1) w1)
      (and (not v0) v1 (not w0) (not w1))
      (and v0 v1 w0 w1))
)

(define-fun a ((v0 Bool) (v1 Bool)) Bool
   (or (not v0) (not v1))
)

(define-fun path1 ((v00 Bool) (v10 Bool) (v01 Bool) (v11
Bool)) Bool
  (and (S0 v00 v10)
    (R v00 v10 v01 v11)
  )
)
```

```
(define-fun path2 ((v00 Bool) (v10 Bool) (v01 Bool) (v11
Bool) (v02 Bool) (v12 Bool)) Bool
  (and (S0 v00 v10)
    (R v00 v10 v01 v11)
    (R v01 v11 v02 v12)
  )
)

; k = 1
;(assert
;  (and (path1 v00 v10 v01 v11)
;    (or (not (a v00 v10))
;      (not (a v01 v11))
;    )
;  )
;)

; k =2
(assert
  (and (path2 v00 v10 v01 v11 v02 v12)
    (or (not (a v00 v10))
      (not (a v01 v11))
      (not (a v02 v12))
    )
  )
)

(check-sat)
(get-model)
```

https://rise4fun.com/Z3

Another Example



Does modulo-8 counter of Chapter 3.5 ever reach 4?

$\neg p = \neg v_0 \wedge \neg v_1 \wedge v_2.$

$S_0(V) = \neg v_0 \wedge \neg v_1 \wedge \neg v_2.$

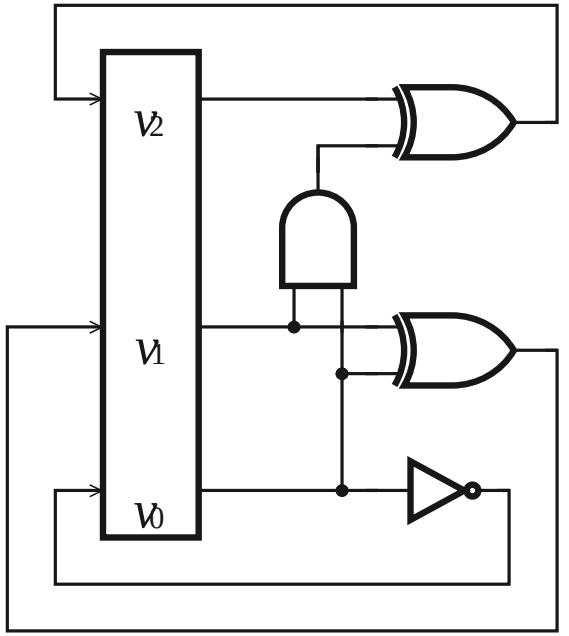
$R(V, V') = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \oplus v_1) \wedge (v_2' \leftrightarrow (v_0 \wedge v_1) \oplus v_2)$.

k =0:

$S_0(v)$ $\neg v_0 \wedge \neg v_1 \wedge \neg v_2 \wedge$

\neg p(v) $\neg v_0 \wedge \neg v_1 \wedge v_2.$

Another Example



Does module-8 counter of Chapter 3.5 ever reach 4?

$\neg p = \neg v_0 \wedge \neg v_1 \wedge v_2.$

$S_0(V) = \neg v_0 \wedge \neg v_1 \wedge \neg v_2.$

$R(V, V') = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \oplus v_1) \wedge (v_2' \leftrightarrow (v_0 \wedge v_1) \oplus v_2)$.

$k = 1$:

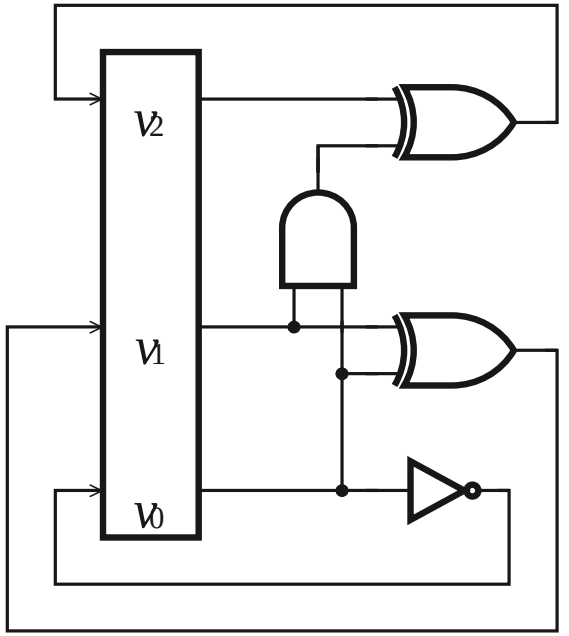
$S_0(v)$ $\neg v_0 \wedge \neg v_1 \wedge \neg v_2 \wedge$

$R(v,v')$ $(v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \oplus v_1) \wedge (v_2' \leftrightarrow (v_0 \wedge v_1) \oplus v_2) \wedge$

$(\neg v_0 \wedge \neg v_1 \wedge v_2 \quad \vee \quad \neg v'_0 \wedge \neg v'_1 \wedge v'_2).$

$\neg p(v)$ $\neg p(v')$

Another Example



Does module-8 counter of Chapter 3.5 ever reach 4?

$\neg p = \neg v_0 \wedge \neg v_1 \wedge v_2.$

$S_0(V) = \neg v_0 \wedge \neg v_1 \wedge \neg v_2.$

$R(V, V') = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \oplus v_1) \wedge (v_2' \leftrightarrow (v_0 \wedge v_1) \oplus v_2)$.

$k = 2$:

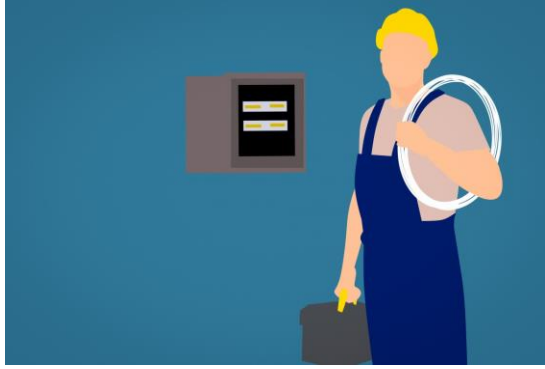
$S_0(v)$ $\neg v_0 \wedge \neg v_1 \wedge \neg v_2 \wedge$

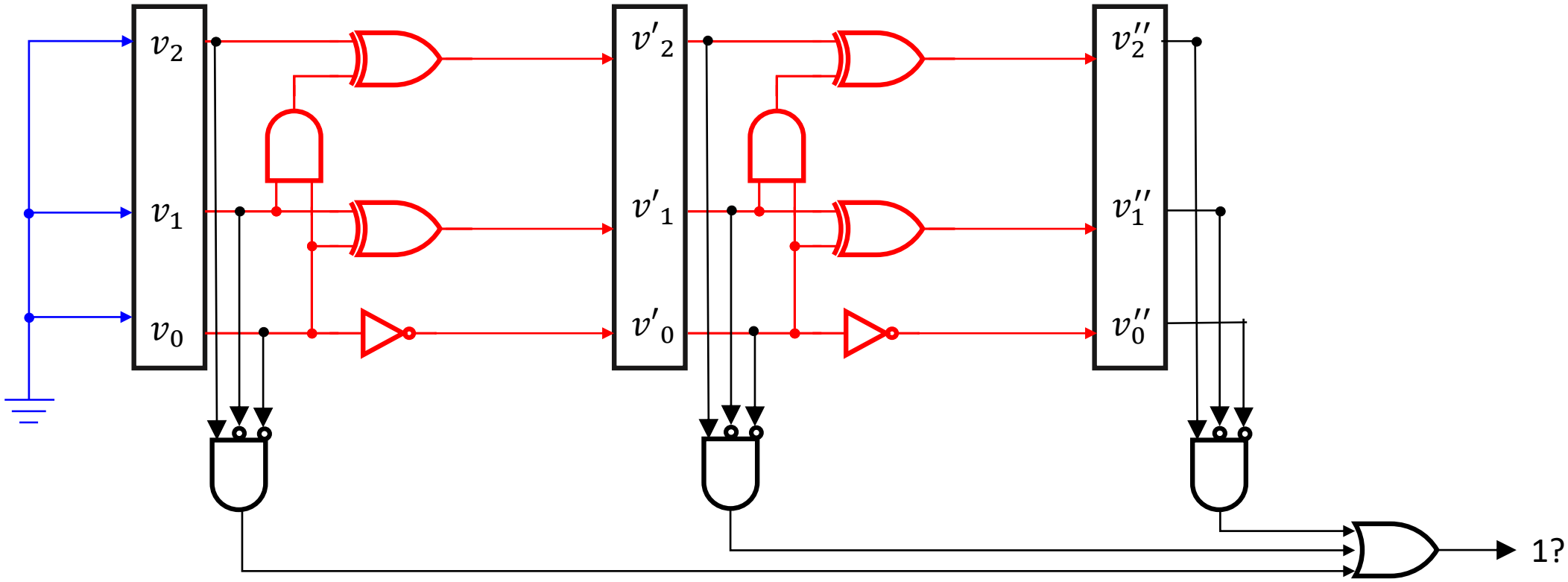
$R(v, v')$ $(v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \oplus v_1) \wedge (v_2' \leftrightarrow (v_0 \wedge v_1) \oplus v_2) \wedge$

$R(v', v'')$ $(v_0'' \leftrightarrow \neg v'_0) \wedge (v''_1 \leftrightarrow v'_0 \oplus v'_1) \wedge (v''_2 \leftrightarrow (v'_0 \wedge v'_1) \oplus v'_2) \wedge$

$(\underbrace{\neg v_0 \wedge \neg v_1 \wedge v_2}_{\neg p(v)} \vee \underbrace{\neg v_0' \wedge \neg v_1' \wedge v_2'}_{\neg p(v')} \vee \underbrace{\neg v''_0 \wedge \neg v''_1 \wedge v''_2}_{\neg p(v'')}.)$

The Electrical Engineer's Viewpoint





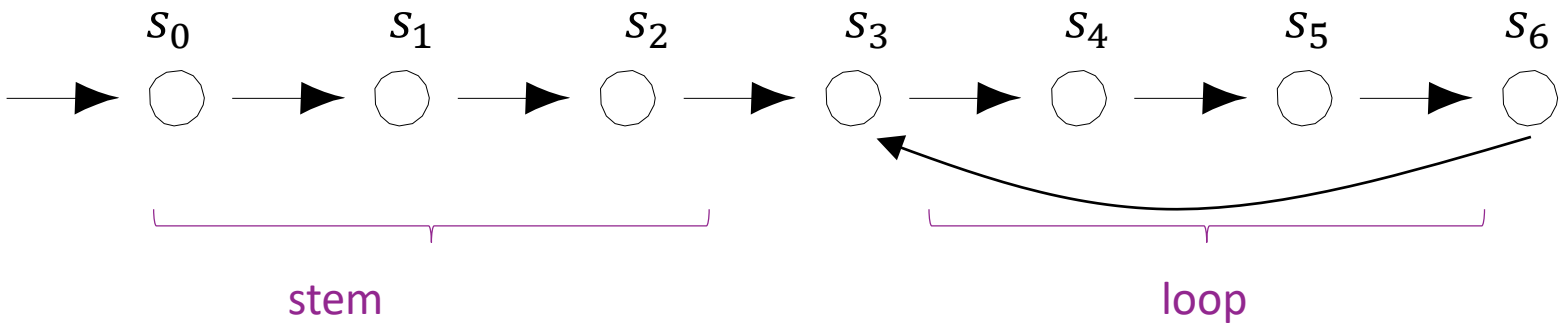
Eventuality Properties

Property

Suppose $\phi = \mathbf{AF}\, p$

Counterexamples fulfill $\mathbf{EG}\, \neg p$

Counterexamples have *Lasso Shape* (See Chapter 4.)



Completeness

BMC finds bugs of length $\leq k$.

Longer counterexamples may exist!

Is there a k big enough to exclude any counterexample?

Def. $M \models_k \phi$: all computations of length k satisfy ϕ.

Completeness threshold: Number CT such that $M \models_{CT} \phi \Rightarrow M \models \phi$

If completeness threshold known, stop BMC when $k = CT$

Ideas for finding CT?

Completeness Threshold

Finding smallest CT is as difficult as model checking!

- Smallest CT is size of shortest counterexample, or 0 if the property is satisfied

Simple values for CT

- Number of state of M is bound on CT
- Diameter (longest simple path between two states) is bound on CT

These are typically really large numbers

Verifying Reachability Properties with k-induction



Mary Sheeran, Koen Claessen, Per Bjesse, 2000

Motivation

- Completeness thresholds usually very large
- Can we **prove** a property with fewer unrollings?
- Idea: Use induction.

Base: Prove $Q(0)$

Induction: Prove $Q(n-1) \Rightarrow Q(n)$

Conclusion: $\forall n. Q(n)$

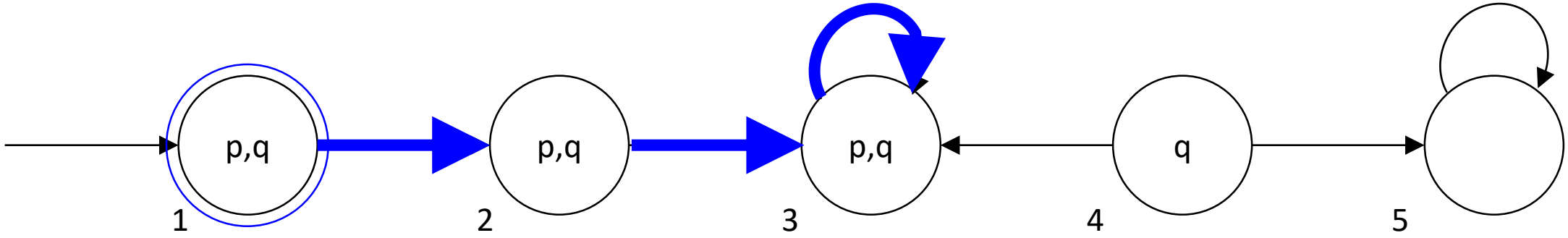
Caveat: Property may be true, but not inductive (see below)

Induction

Let's prove $\mathbf{AG}\ p$ on the following structure.

Take arbitrary path π

- **Base case:** $\pi(0) \vDash p$ true: $q_1 \vDash p$
- **Induction:** if $\pi(n-1) \vDash p$ then $\pi(n) \vDash p$ true: any successor of a p-state is a p-state
- **Conclusion**: for any path π we have $\forall n.\, \pi(n) \vDash p$

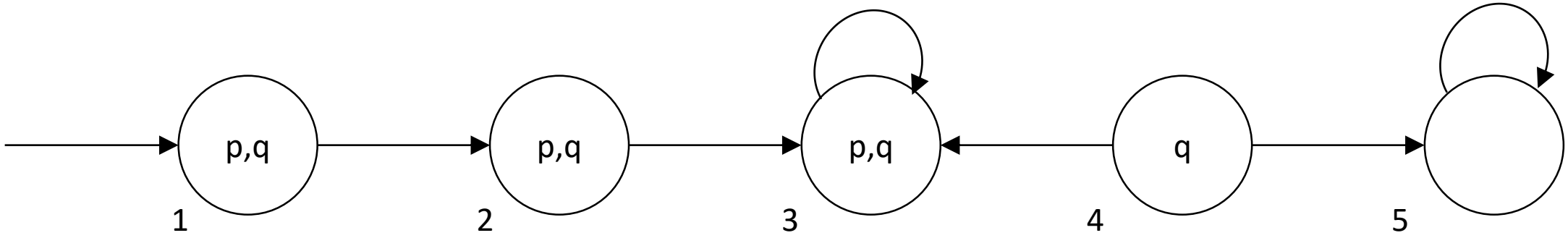


Satisfiability

Let's prove $\mathrm{AG}\ p$ on the following structure. How can these properties be violated?

Take arbitrary path π

- **Base case:** $\pi(0) \vDash p$ $S_o(s) \wedge \neg \mathrm{p}(s)$ Unsatisfiable
- **Induction:** if $\pi(n-1) \vDash p$ then $\pi(n) \vDash p$ $p(s) \wedge R(s,s') \wedge \neg p(s')$ Unsatisfiable
- **Conclusion**: for any path π we have $\forall n.\, \pi(n) \vDash p$

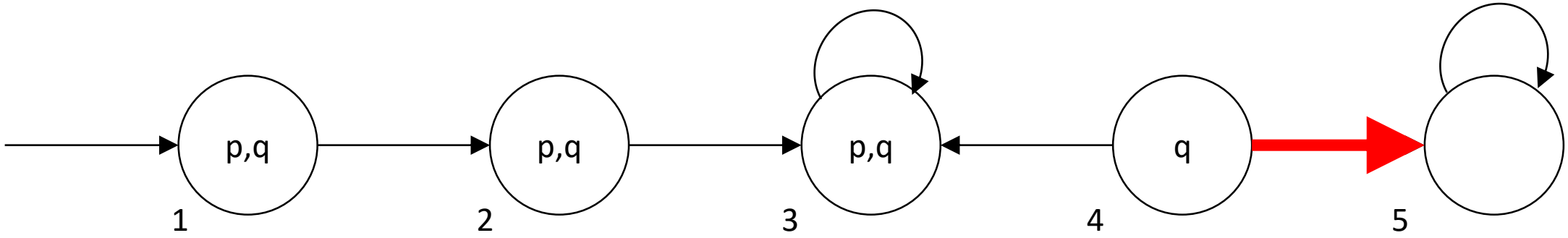


A Problem

Let's prove $\mathbf{AG}\ q$ on the following structure.

Take arbitrary path π

- **Base case:** $\pi(0) \vDash \mathrm{q}$
- **Induction:** if $\pi(n-1) \vDash q$ then $\pi(n) \vDash q$ **not true!**
- ~~**Conclusion**: for any path π we have $\forall n.\pi(n) \vDash q$~~ **not all true properties are inductive**



k-induction

Base:

Induction:

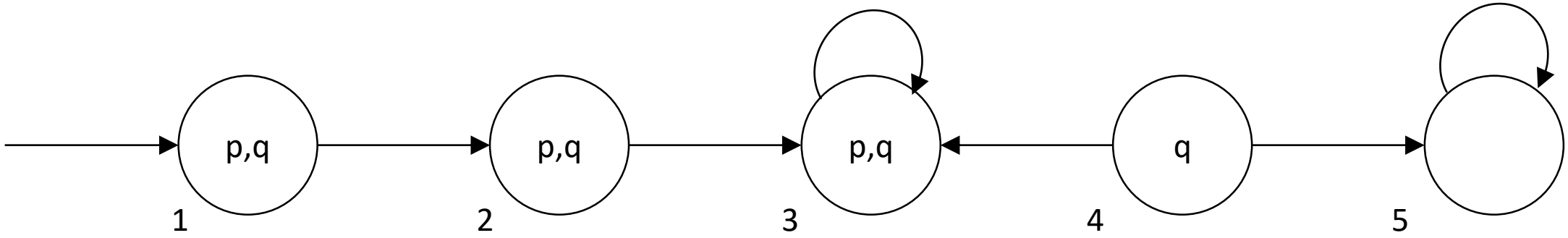
Conclusion: $\forall n.\, Q(n)$

In our setting:

Base. all paths of length *k* from S_0 are labeled *q*

Induction. all paths of length *k* labeled with all *q*s are followed by a *q*

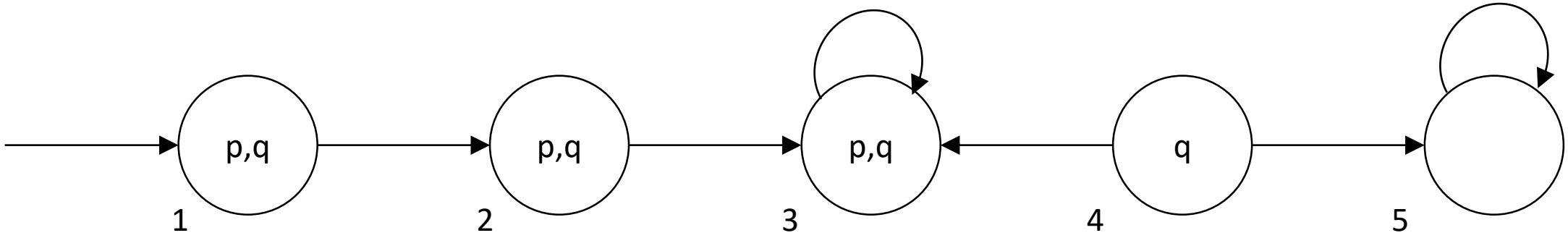
Conclusion. All paths from S_0 are labeled *q*



k-induction

Base: Prove $Q(0) \wedge \cdots \wedge \mathrm{Q}(k-1)$
Induction: Prove $\mathrm{Q}(\mathrm{n}-\mathrm{k}) \wedge \cdots \wedge Q(n-1) \Rightarrow Q(n)$
Conclusion: $\forall n.\, Q(n)$

In our setting:
Base. all paths of length *k* from S_0 are labeled *q*
Induction. all paths of length *k* labeled with all *q*s are followed by a *q*
Conclusion. All paths from S_0 are labeled *q*



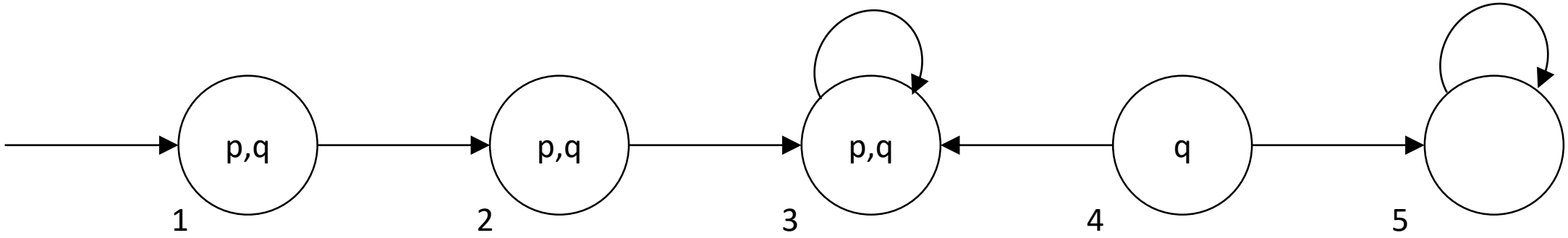
Prove AG q using 2-induction

Base: Consider all paths of length 2 from q_1: $q_1 \vDash q$ and $q_2 \vDash q$.

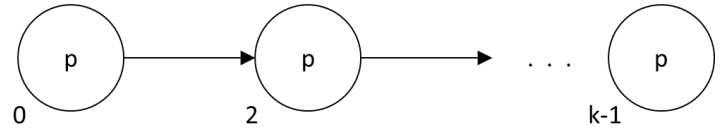
Induction: Do all successors of paths of length 2 labeled q, q fulfill q?

- (q_1, q_2)
- (q_2, q_3)
- (q_3, q_4)
- (q_4, q_4)

Conclusion: for any path π we have $\forall n. \pi(n) \vDash p$



k-induction as Satisfiability



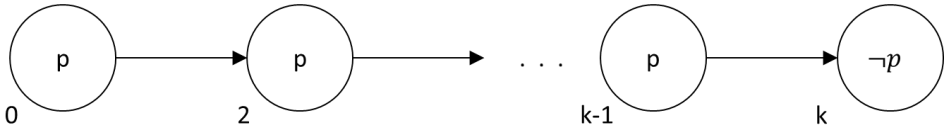
Base. all paths of length *k* from S_0 are labeled *p*

This is BMC! $S_0(s_0) \wedge \bigwedge_{i=0}^{k-2} R(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k-1} \neg p(s_i)$

Induction. all paths of length *k* labeled with all *p*'s are followed by a *p*

$$\bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge \bigwedge_{i=0}^{k-1} p(s_i) \wedge \neg p(s_k)$$

Formula satisfiable iff there is a counterexample



k-induction

while(k=0; ; k++){

build BMC formula ϕ

if ϕ SAT return “bug!”

build induction formula ψ

if ϕ UNSAT return “correct!”

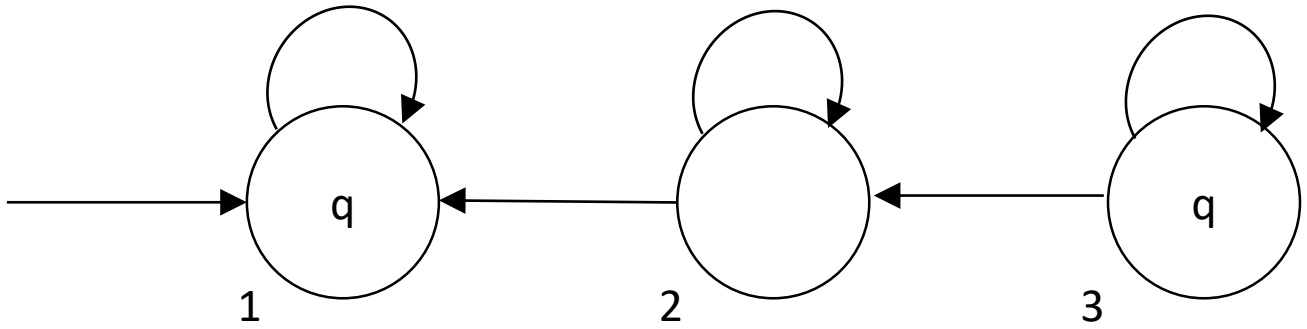
}

k-induction is not Complete

System satisfies $\mathbf{AG}\ q$, but induction step fails for any *k*

Base. all paths of length *k* from S_0 are labeled *q*

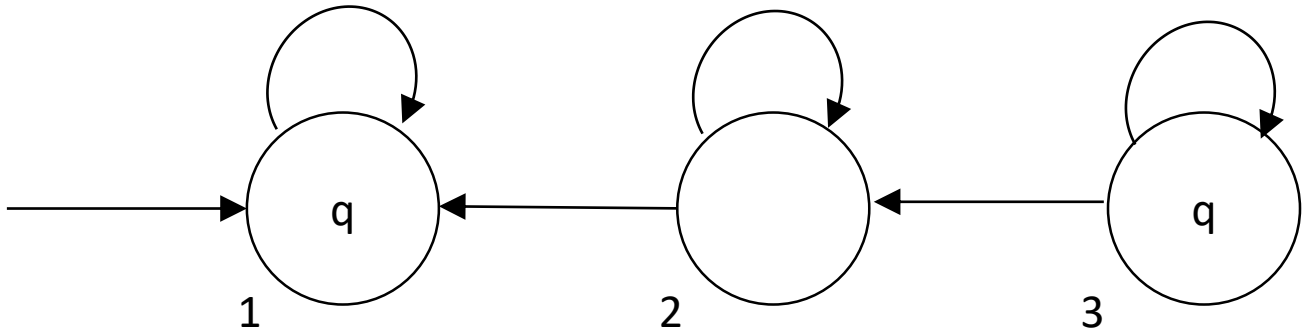
Induction. all paths of length *k* labeled with all *q*s are followed by a *q*. *FALSE*



Making k-induction Complete

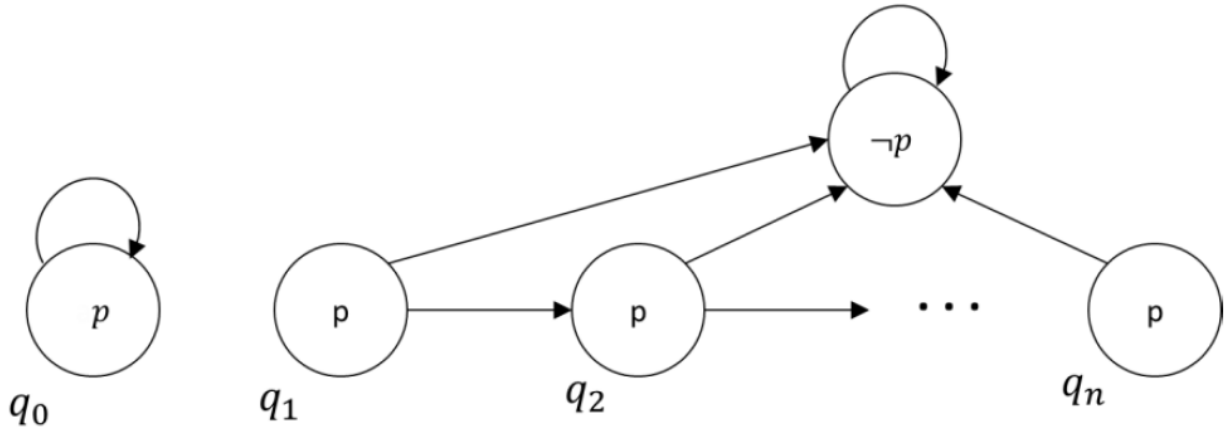
Induction. all noncyclic paths of length *k* labeled with all *q*s are followed by a *q*

$$\bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge \bigwedge_{i=0}^{k-1} p(s_i) \wedge \neg p(s_k) \wedge \bigwedge_{i=0}^{k-1} \bigwedge_{j=i+1}^{k} s_i \neq s_j$$



HW2

Consider the following synchronous Kripke structure K.



We wish to prove that p is always true.

Task 2a. [5 points]

Suppose that q_1 is the initial state. Suppose you are given formulas R, S_0, and p for the transition relation, the initial states and the property, resp.

- What is the smallest *k* such that BMC finds a counterexample?
- Show the BMC formula, using R, S_0, and p.
- Is the formula satisfiable? Explain.

Task 2c. [5 points]

Suppose that q_0 is the initial state. The new formula for the initial states is S'_0.

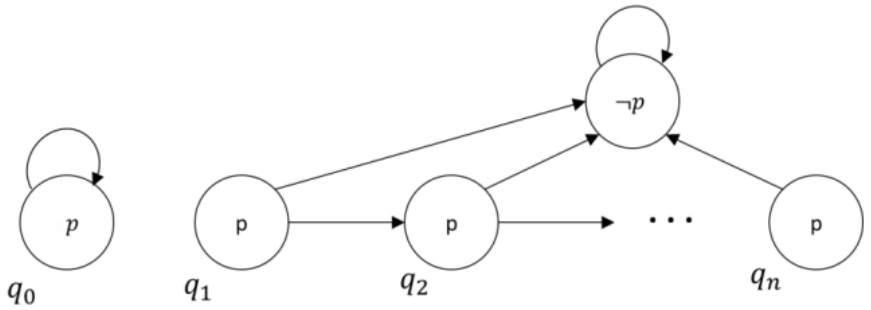
- What is the smallest *k* such that k-induction can prove the property correct?
- Suppose *n=2*. Choose an appropriate *k* and show the k-induction formula, using R, S'_0, and p.
- Is the formula satisfiable? Explain.

Note to myself

- There is a BASE formula and an INDUCTION formula. You need both to do k-induction. That was unclear in the home work.h

Model Checking with Inductive Invariants

Consider the following synchronous Kripke structure K.



We wish to prove that *p* is always true.

Task 2a. [5 points]

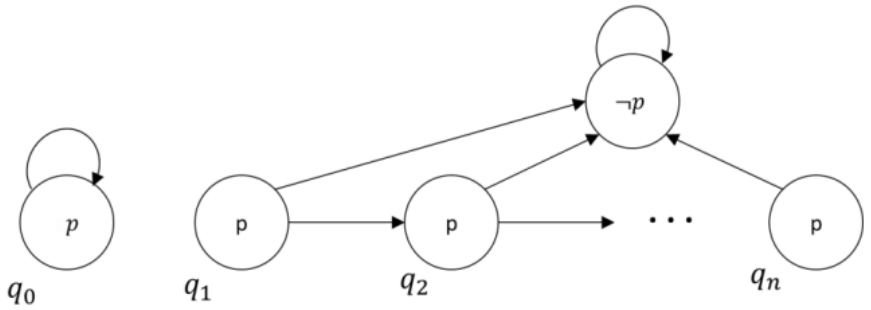
Suppose that q_1 is the initial state. Suppose you are given formulas R, S_0, and p for the transition relation, the initial states and the property, resp.

- What is the smallest *k* such that BMC finds a counterexample?
- Show the BMC formula, using R, S_0, and p.
- Is the formula satisfiable? Explain.

$$path_k(s_0, \ldots, s_k) = S_0(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1})$$

$$path_k(s_0, \ldots, s_k) \wedge \bigvee_{i=0}^{k} \neg p(s_i)$$

Consider the following synchronous Kripke structure K.



We wish to prove that *p* is always true.

Task 2c. [5 points]

Suppose that q_0 is the initial state. The new formula for the initial states is S'_0.

- What is the smallest *k* such that k-induction can prove the property correct?
- Suppose *n=2*. Choose an appropriate *k* and show the k-induction formula, using R, S'_0, and p.
- Is the formula satisfiable? Explain.

$$S_0(s_0) \wedge \bigwedge_{i=0}^{k-2} R(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k-1} \neg p(s_i) \qquad \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge \bigwedge_{i=0}^{k-1} p(s_i) \wedge \neg p(s_k)$$

Homework Results

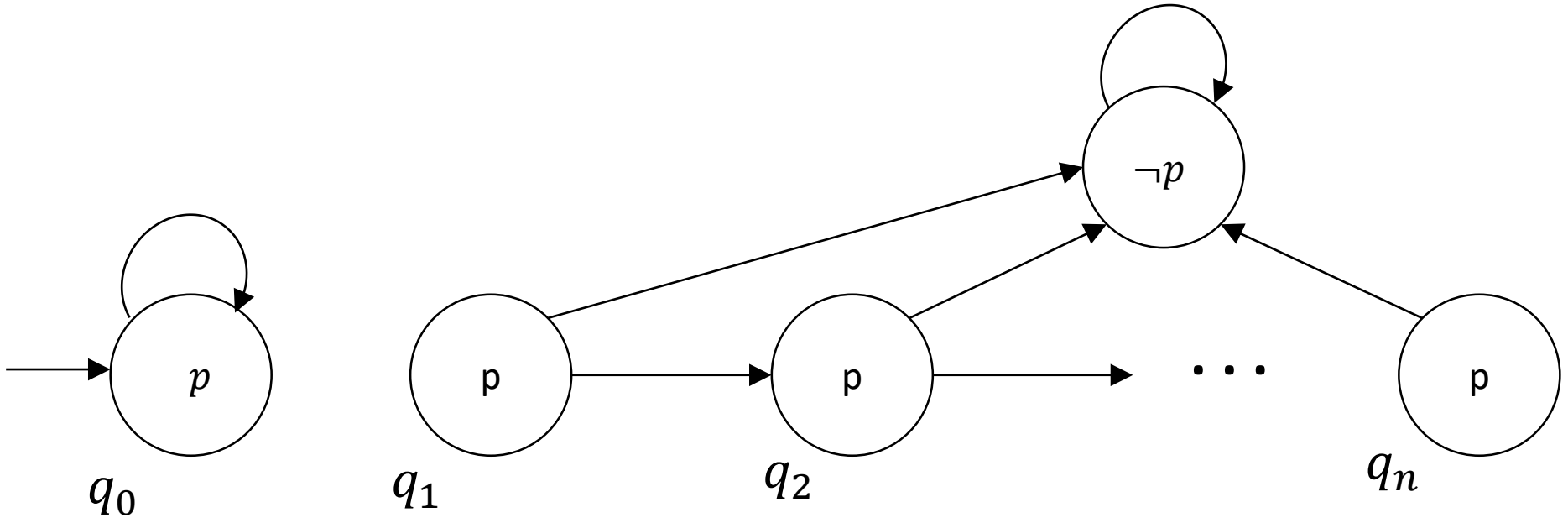
- Find the results here: https://cloud.tugraz.at/index.php/s/zeEgt8ptcRQCXEW
- Confused? Write an email to modelchecking@iaik.

Problems with *k*-induction

Problem: Sometimes *k* is very large

In the following machine, you need $k = n + 1$ to prove $\mathbf{AG}\, p$.

Idea: Automatically find better inductive invariants.



Inductive Invariant

Remember $postimage(Q) = \{ s' \mid \exists s.\, R(s, s') \}$ (see Chapter 5).

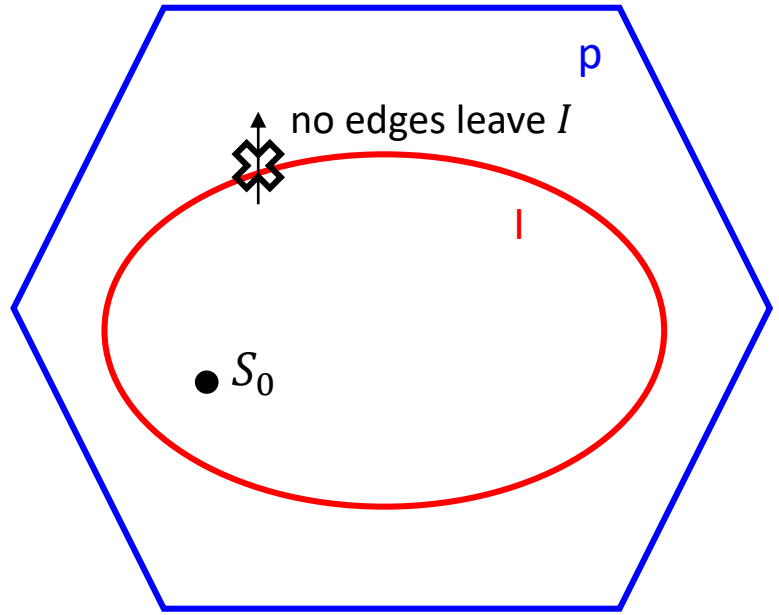
Definition. $I \subseteq S$ is an inductive invariant for AG p if

1. $S_0 \subseteq I$
2. $postimage(I) \subseteq I$
3. $\forall s \in I.\, s \vDash p$

If there is an inductive invariant for AG p, then AG p holds.

In formulas:

1. $S_0 \rightarrow I$
2. $I \wedge R \rightarrow I'$
3. $I \rightarrow p$



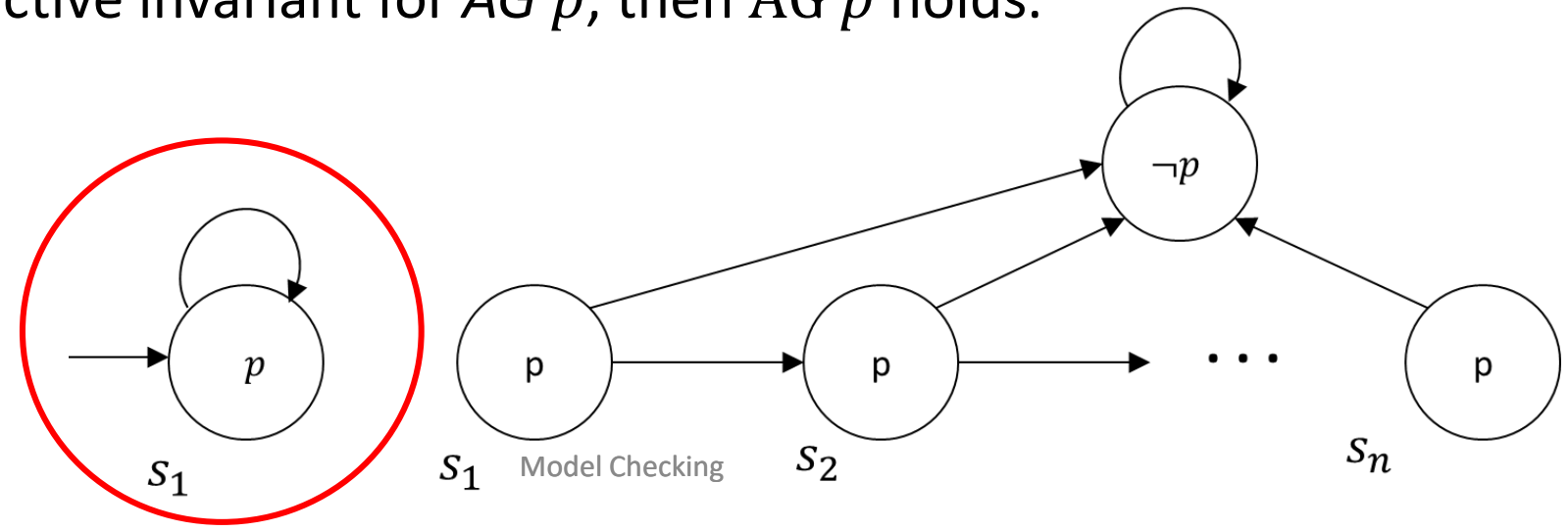
Inductive Invariant

Remember $postimage(Q) = \{ s' \mid \exists s. R(s, s') \}$ (see Chapter 5).

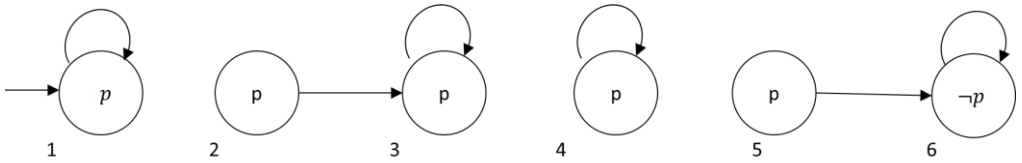
Definition. $I \subseteq S$ is an inductive invariant for AG p if

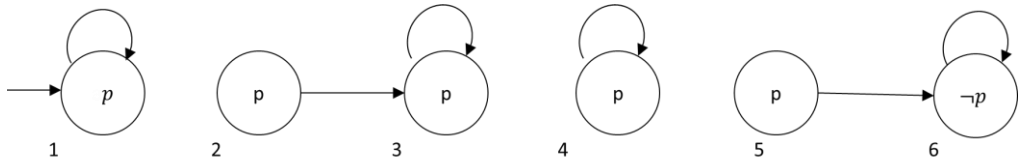
1. $S_0 \subseteq I$
2. $postimage(I) \subseteq I$
3. $\forall s \in I. s \vDash p$

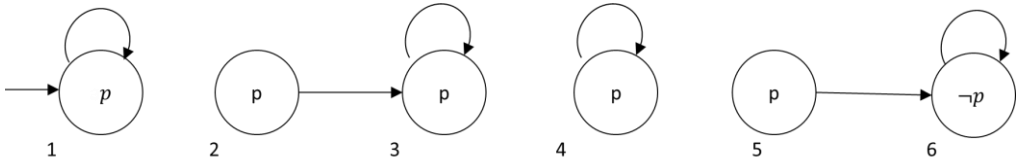
If there is an inductive invariant for *AG* p, then AG p holds.

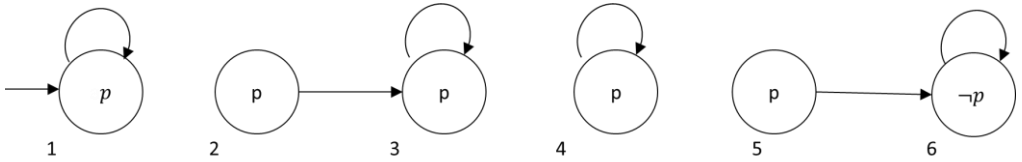


Multiple Invariants

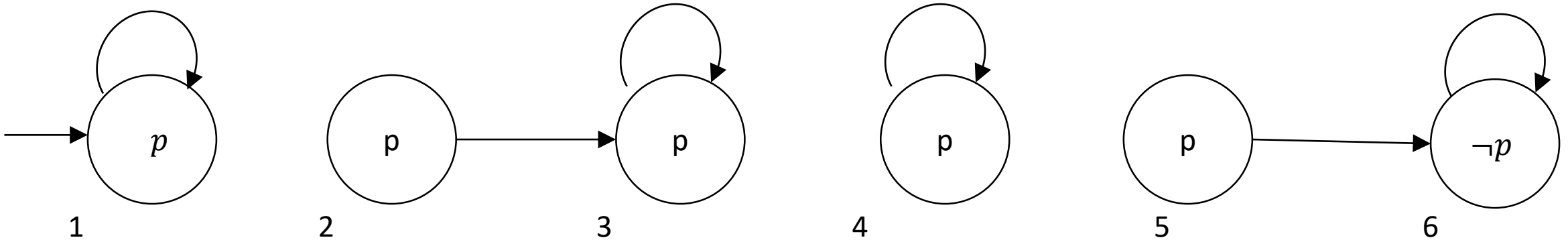








Strongest & Weakest Invariant



Smallest (strongest) invariant is reachable state
Largest (weakest) invariant is states that cannot reach $\neg p$

Model Checking with Craig Interpolants

Ken McMillan, 2003

2010 CAV Award: “has significantly influenced both academic research and industrial practice, and has dramatically changed the scale of systems that can be analyzed by model checking.”

Kenneth McMillan

Interpolants as Inductive Invariants

- BMC finds bugs (and absence of bugs up to k steps)
- How to Show Correctness?
  - k-induction
  - Interpolants

- Find Interpolants I such that
  - States reachable in k steps are in I
  - no bad states are in I
- Interpolants are (good) overapproximation of post-image computation

Interpolant

William Craig, 1957

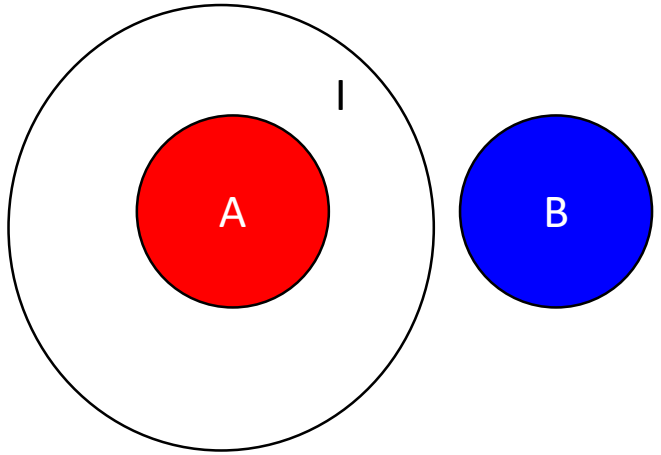
Definition. Given formulas A, B such that $A \wedge B = \bot$, an interpolant is a formula I such that

1. $A \rightarrow I$
2. $I \wedge B \equiv \bot$
3. I only uses symbols that occur both in A and in B

Example. Let

$A = (a_1 \vee \neg a_2) \wedge (\neg a_1 \vee \neg a_3) \wedge a_2$,

$B = (\neg a_2 \vee a_3) \wedge (a_2 \vee a_4) \wedge \neg a_4$.

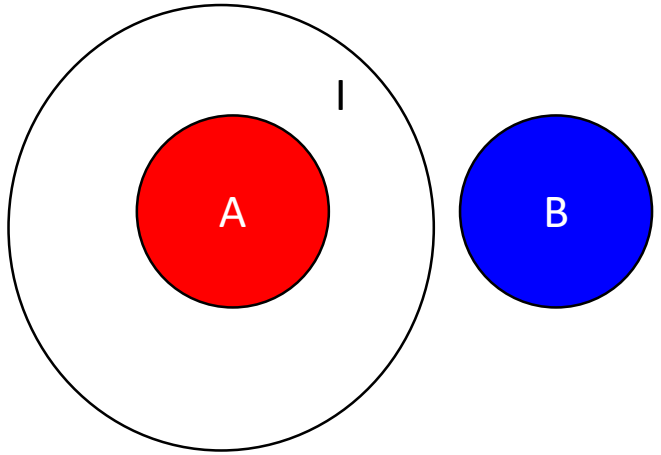


Interpolant

William Craig, 1957

Definition. Given formulas A, B such that $A \wedge B = \bot$, an interpolant is a formula I such that

1. $A \rightarrow I$
2. $I \wedge B \equiv \bot$
3. I only uses symbols that occur both in A and in B



Example. Let

$A = (a_1 \vee \neg a_2) \wedge (\neg a_1 \vee \neg a_3) \wedge a_2$,

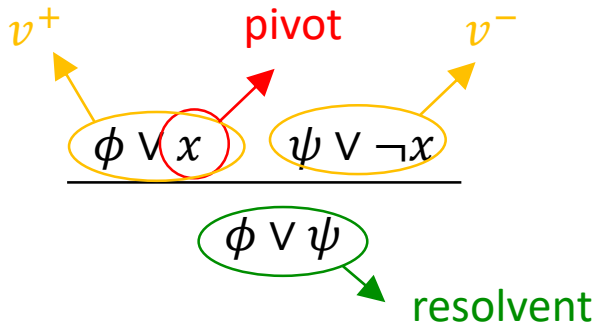
$B = (\neg a_2 \vee a_3) \wedge (a_2 \vee a_4) \wedge \neg a_4$.

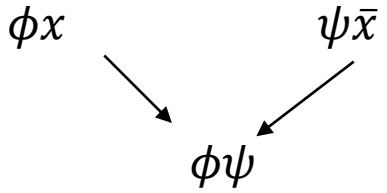
$A \wedge B$ is not satisfiable.

$\neg a_3 \wedge a_2$ is an interpolant:

1. $\big((a_1 \vee \neg a_2) \wedge (\neg a_1 \vee \neg a_3) \wedge a_2\big) \rightarrow (\neg a_3 \wedge a_2)$
2. $(\neg a_3 \wedge a_2) \wedge \big((\neg a_2 \vee a_3) \wedge (a_2 \vee a_4) \wedge \neg a_4\big) \equiv \bot$
3. a_2 and a_3 occur in A and in B

Resolution (Chap 9)





Interpolants from Resolution Proofs

For clause C, $C|B$ is obtained by removing literals not in B

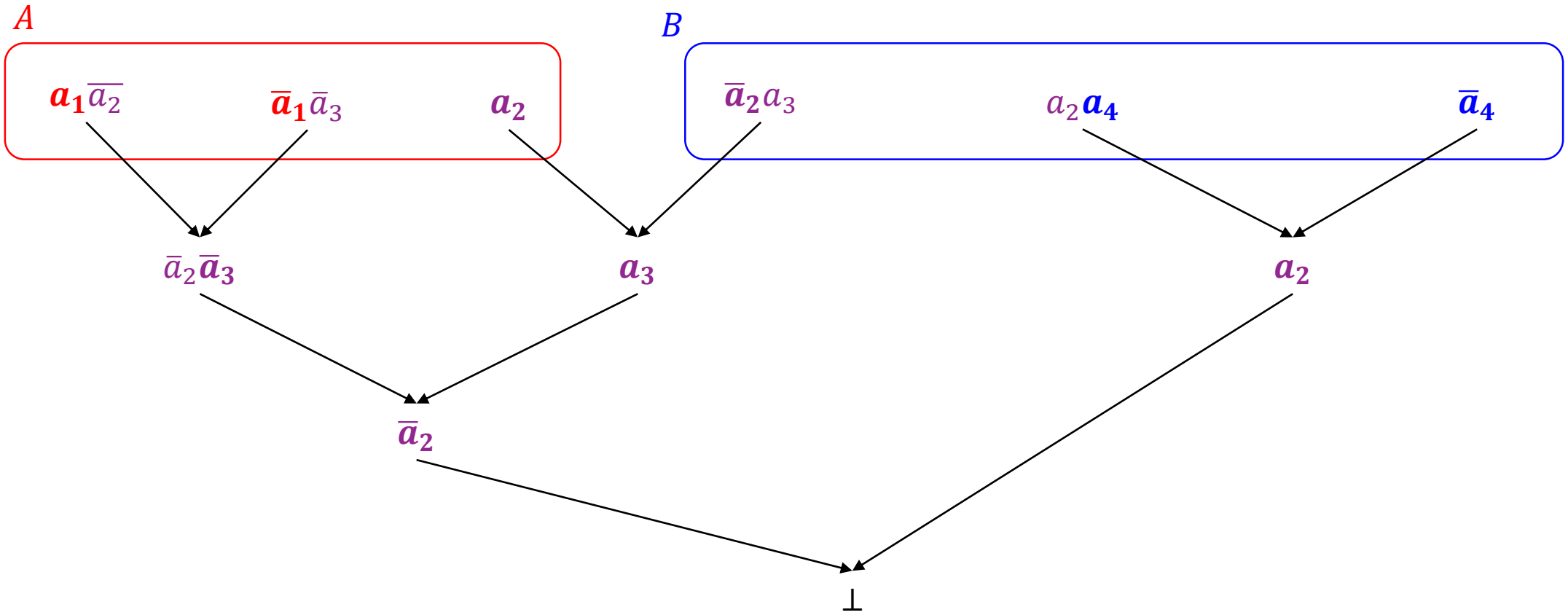
Algorithm. Go through resolution proof top-down.

1. If leaf v is labeled $C \in A$, then $Itp(v) = C|B$
2. If leaf v is labeled $C \in B$, then $Itp(v) = \top$
3. If node v has pivot variable $x \in B$ then $Itp(v) = Itp(v^+) \wedge Itp(v^-)$
4. If node v has pivot variable $x \notin B$ then $Itp(v) = Itp(v^+) \vee Itp(v^-)$

Interpolation Example

Algorithm. Go through resolution proof top-down.

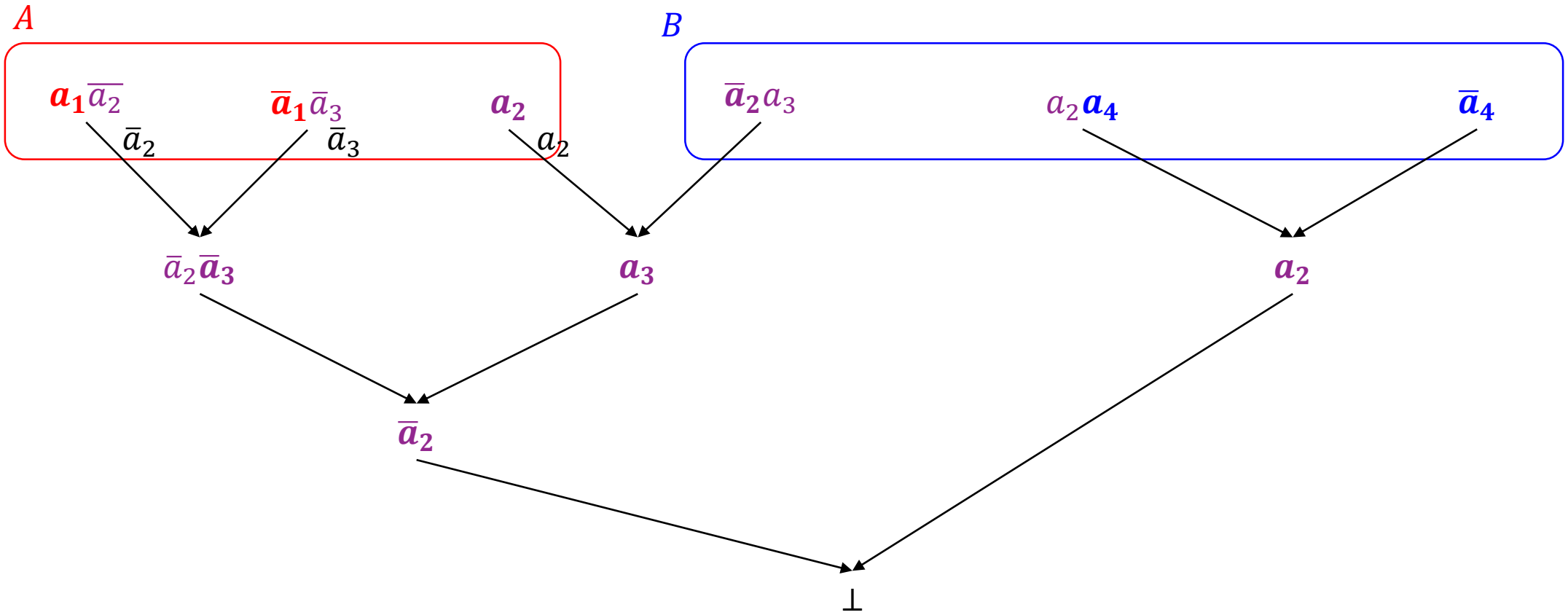
1. If leaf v is labeled $C \in A$, then $Itp(v) = C|B$
2. If leaf v is labeled $C \in B$, then $Itp(v) = \top$
3. If node v has pivot variable $x \in B$ then $Itp(v) = Itp(v^+) \wedge Itp(v^-)$
4. If node v has pivot variable $x \notin B$ then $Itp(v) = Itp(v^+) \vee Itp(v^-)$



Interpolation Example

Algorithm. Go through resolution proof top-down.

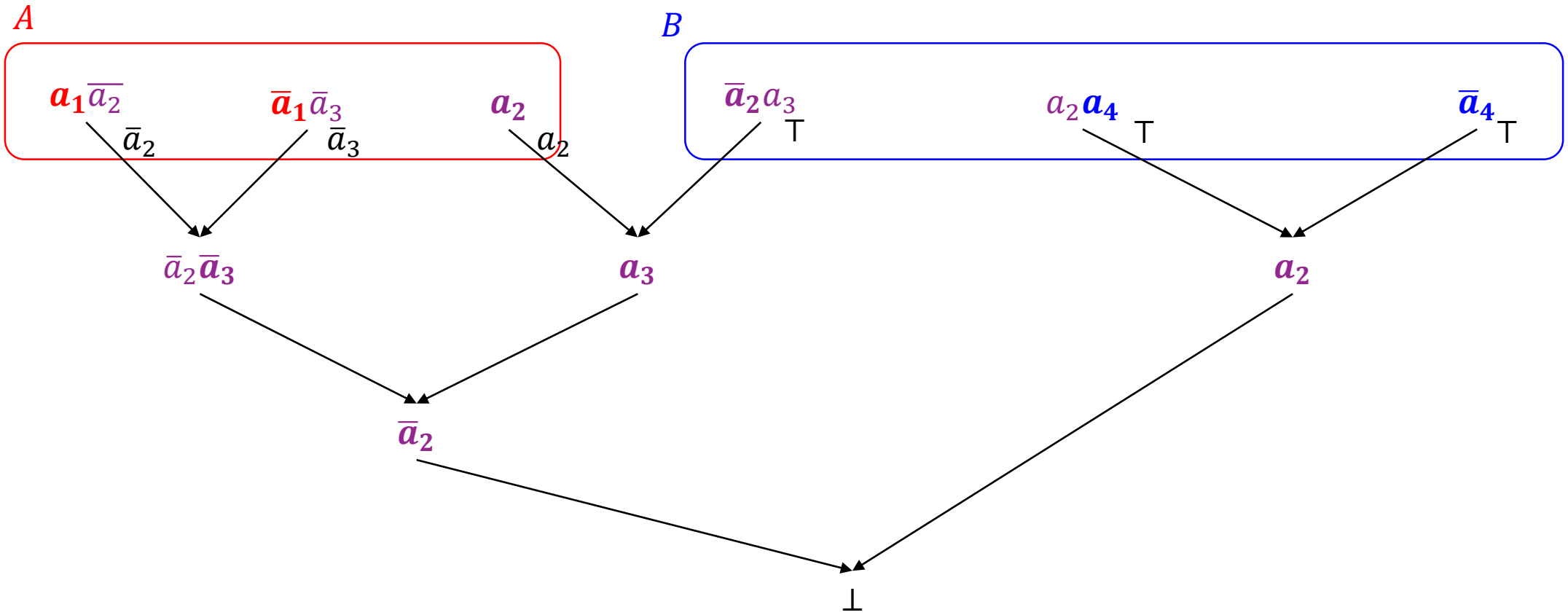
1. If leaf v is labeled $C \in A$, then $Itp(v) = C|B$
2. If leaf v is labeled $C \in B$, then $Itp(v) = \top$
3. If node v has pivot variable $x \in B$ then $Itp(v) = Itp(v^+) \wedge Itp(v^-)$
4. If node v has pivot variable $x \notin B$ then $Itp(v) = Itp(v^+) \vee Itp(v^-)$



Interpolation Example

Algorithm. Go through resolution proof top-down.

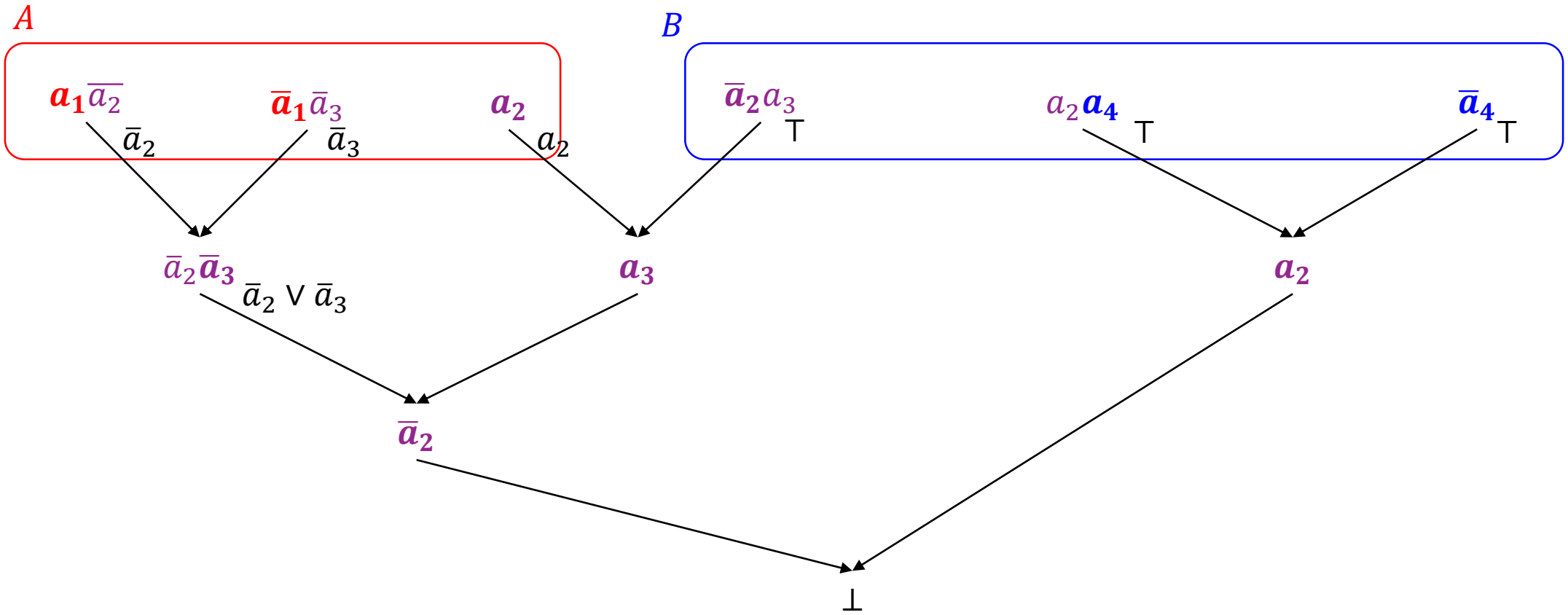
1. If leaf v is labeled $C \in A$, then $Itp(v) = C|B$
2. If leaf v is labeled $C \in B$, then $Itp(v) = \top$
3. If node v has pivot variable $x \in B$ then $Itp(v) = Itp(v^+) \wedge Itp(v^-)$
4. If node v has pivot variable $x \notin B$ then $Itp(v) = Itp(v^+) \vee Itp(v^-)$



Interpolation Example

Algorithm. Go through resolution proof top-down.

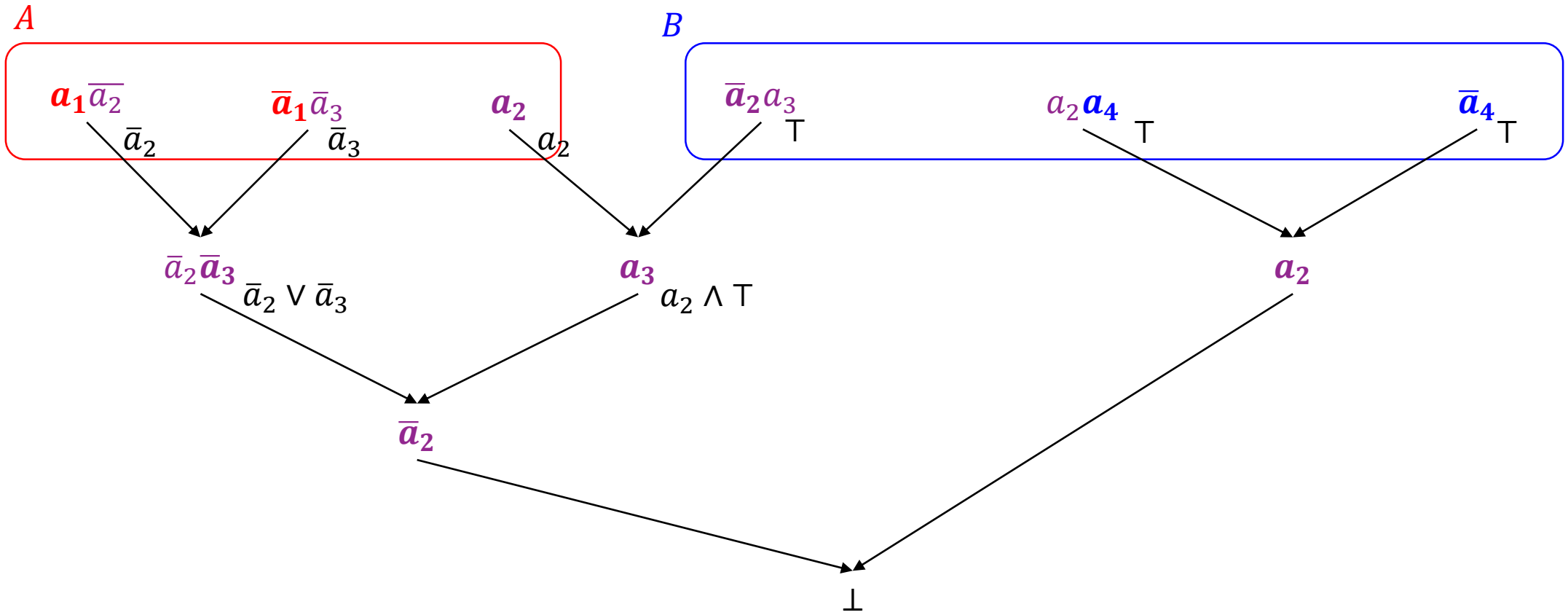
1. If leaf v is labeled $C \in A$, then $Itp(v) = C|B$
2. If leaf v is labeled $C \in B$, then $Itp(v) = \top$
3. If node v has pivot variable $x \in B$ then $Itp(v) = Itp(v^+) \wedge Itp(v^-)$
4. If node v has pivot variable $x \notin B$ then $Itp(v) = Itp(v^+) \vee Itp(v^-)$



Interpolation Example

Algorithm. Go through resolution proof top-down.

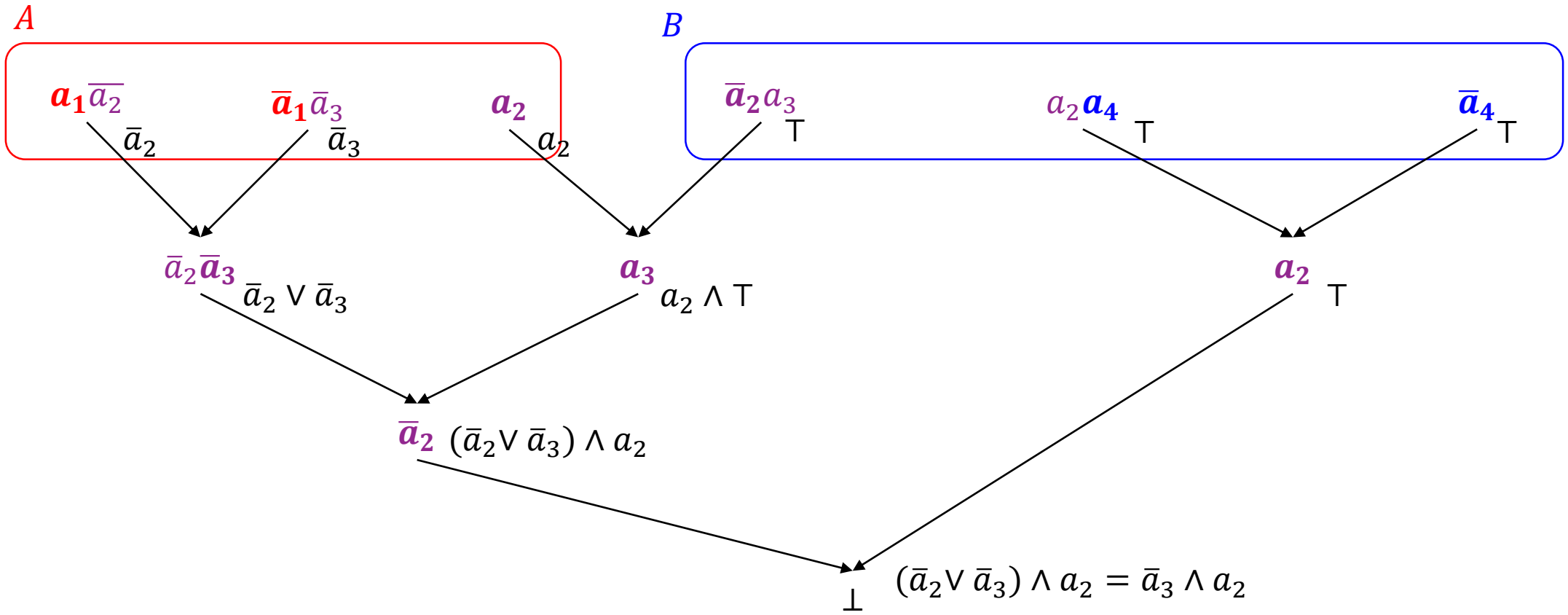
1. If leaf v is labeled $C \in A$, then $Itp(v) = C|B$
2. If leaf v is labeled $C \in B$, then $Itp(v) = \top$
3. If node v has pivot variable $x \in B$ then $Itp(v) = Itp(v^+) \wedge Itp(v^-)$
4. If node v has pivot variable $x \notin B$ then $Itp(v) = Itp(v^+) \vee Itp(v^-)$



Interpolation Example

Algorithm. Go through resolution proof top-down.

1. If leaf v is labeled $C \in A$, then $Itp(v) = C|B$
2. If leaf v is labeled $C \in B$, then $Itp(v) = \top$
3. If node v has pivot variable $x \in B$ then $Itp(v) = Itp(v^+) \wedge Itp(v^-)$
4. If node v has pivot variable $x \notin B$ then $Itp(v) = Itp(v^+) \vee Itp(v^-)$



Reachability Checking with Interpolation

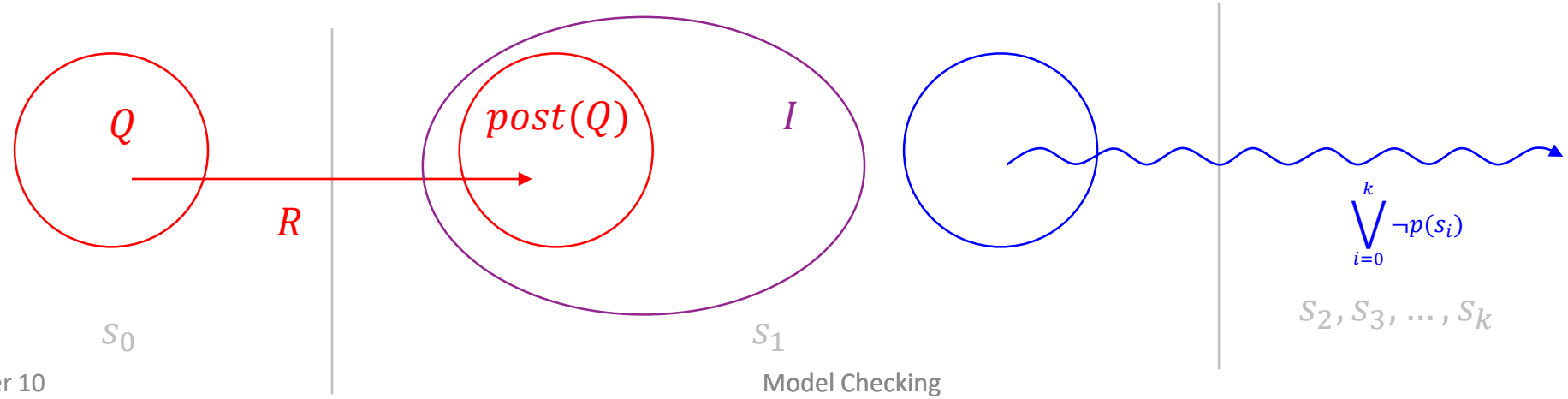
Recall BMC check for $\neg \mathbf{AG} p$:

$S_0(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg p(s_i)$.

Start from Q such that $Q \vDash p$

$\phi = Q(s_0) \wedge R(s_0, s_1) \wedge \bigwedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=1}^{k} \neg p(s_i)$.

Suppose ϕ unsatisfiable, $I(s_1)$ is an interpolant



Reachability Checking with Interpolation

Recall BMC check for $\neg \mathbf{AG} p$:

$S_0(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg p(s_i)$.

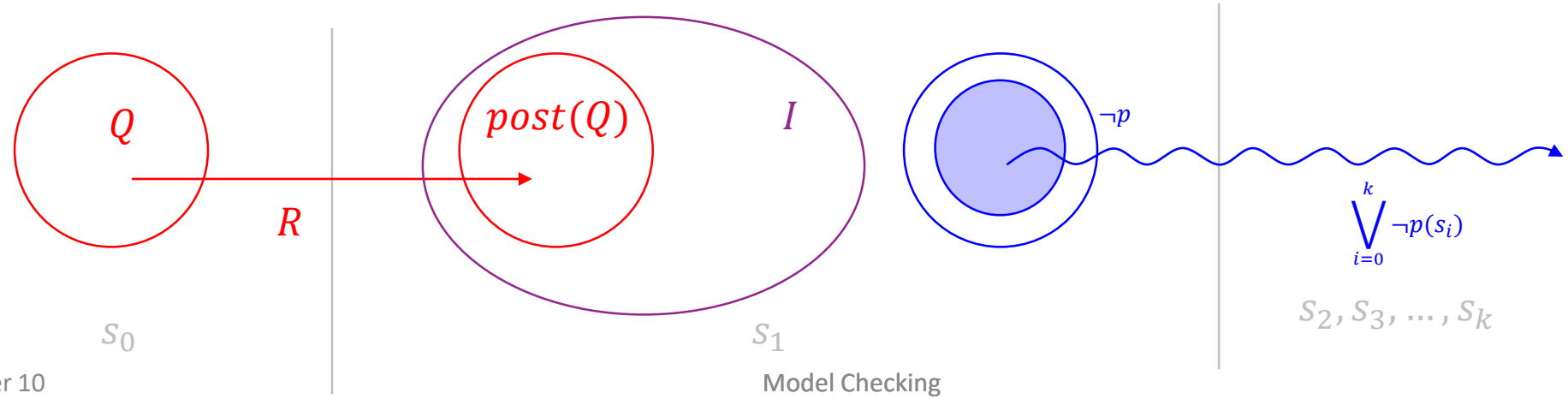
Start from Q such that $Q \models p$

$\phi = Q(s_0) \wedge R(s_0, s_1) \wedge \bigwedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=1}^{k} \neg p(s_i)$.

Suppose ϕ unsatisfiable, $I(s_1)$ is an interpolant.

Note 1: $\neg p(s_1) \rightarrow B$
so $I(s_1) \wedge \neg p(s_1) = \bot$

Note 2: $I \supseteq post(Q)$



Interpolant Reachability Idea

$\phi = Q(s_0) \wedge R(s_0, s_1) \wedge \bigwedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg p(s_i)$.

1. Start with $Q = S_0$
2. If ϕ is satisfiable, $\neg p$ is **reachable**
3. If not, set Q to I
4. If I remains unchanged, p **cannot be reached** (Interpolants are approximation to post-image)
5. If ϕ is eventually satisfiable, increase k to increase precision of approximation.

Procedure terminates when k is diameter of system (or earlier!)

Algorithm

procedure CraigReachability(model M, $p \in AP$)
  if $S_0 \wedge \neg$p is SAT return "$M \not\models \mathrm{AG}\, p$ ";
  k := 1;
  $Q := S_0(s_0)$;
  while true **do**
  $A := Q(s_0) \wedge R(s_0, s_1)$;
  $B := \bigvee_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=1}^{k} \neg p(s_i)$;
  if $A \wedge B$ is SAT **then** // $\neg p$ can be reached from Q
    if $Q = S_0$ then return "$M \not\models \mathrm{AG}\, p$"; // $\neg p$ can be reached from S_0
    Increase k // Not sure if path to $\neg p$ is real. Increase precision
    $Q := S_0(s_0)$;
  else
    compute interpolant I for A and B;
    If $I(s_0) == Q$ then return "$M \models \mathrm{AG}\, p$"; // Reached the fixpoint of overapproximated reachability?
    $Q := Q \vee I(s_0)$; // Another step of overapproximated reachability?
  end if
  end while
end procedure

10.4.4 Correctness

If CraigReachability returns "$\boldsymbol{M \models AG\ p}$" then $\boldsymbol{M \models AG\ p}$

Let Q_i denote Q at iteration i. For all i, $Q_i \leftarrow postimage^i(Q_0)$. If $I \rightarrow Q_i$, we have reached a fixed point $Q^* = Q_i$ so $Q^* \leftarrow postimage^*(Q_0)$. Now because $Q_i \wedge \neg p = \bot$, we have $postimage^*(Q_0) \wedge \neg p = \bot$.

If CraigReachability returns "$\boldsymbol{M \not\models AG\ p}$" then $\boldsymbol{M \not\models AG\ p}$

$A \wedge B$ encodes a path from Q_0 to $\neg p$.

CraigReachability terminates

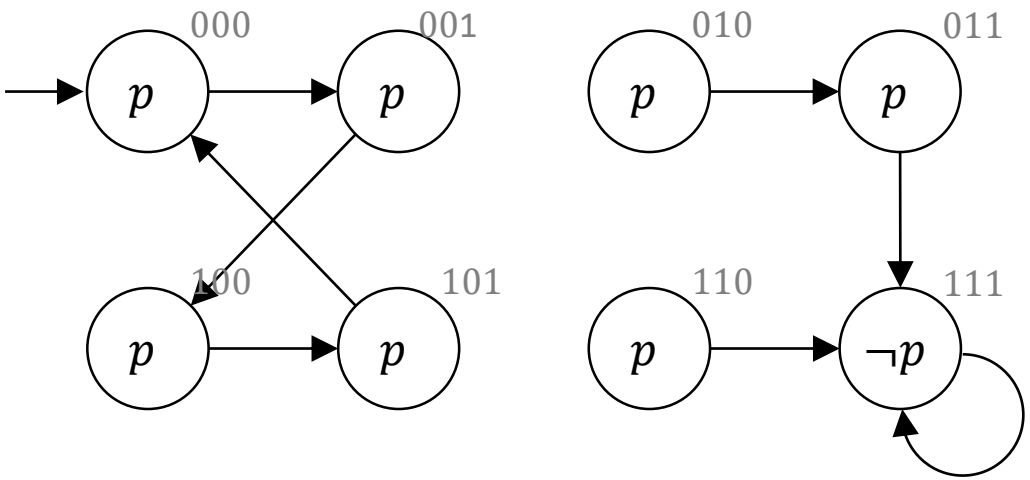
Note that k increases.

If $M \not\models AG\ p$, there is a path of length l to $\neg p$ and we will find it when $l = k$.

Suppose $M \models AG\ p$. If k is the diameter of the graph, no I and thus no Q_i can contain a state that reaches $\neg p$. Thus, $A \wedge B$ is never SAT and the algorithm terminates because the Q_i cannot grow forever.

$x_1x_2x_3$

Example $\mathbf{AG}\ p$



$$\phi = Q(s_0) \wedge R(s_0, s_1) \wedge \bigwedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=1}^{k} \neg p(s_i).$$

if $A \wedge B$ is SAT **then**

  if $Q = S_0$ then return "$M \not\models \mathrm{AG}\, p$";

  increase k

$Q := S_0(s_0)$;

else

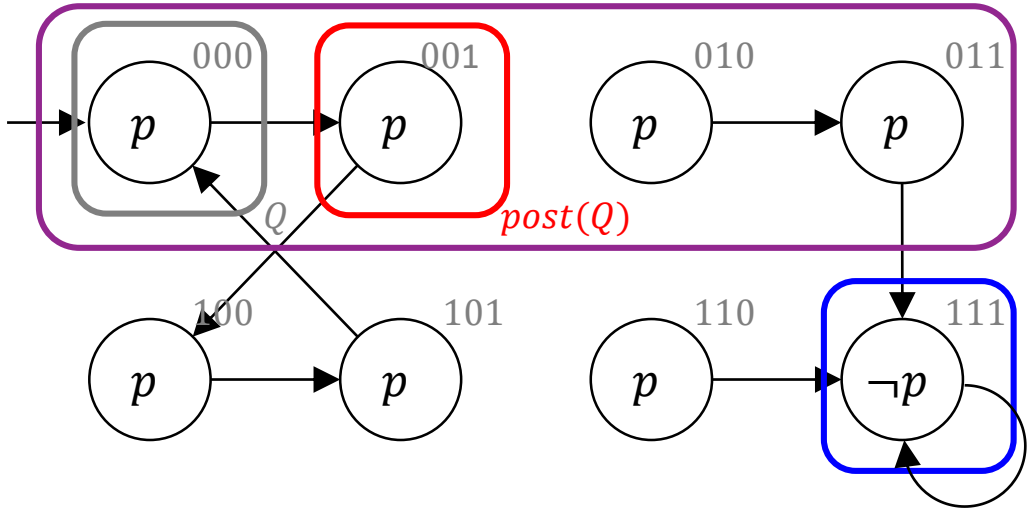
  compute interpolant I for A and B;

  if $I(s_0) \rightarrow Q$ then return "$M \models \mathrm{AG}\, p$";

$Q := Q \vee I(s_0)$;

$x_1x_2x_3$

Example $\mathbf{AG}\ p$



$\phi = Q(s_0) \wedge R(s_0, s_1) \wedge$
$\wedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \vee_{i=1}^{k} \neg p(s_i)$.

$\boldsymbol{k = 1}$.

$Q = \neg x_1 \wedge \neg x_2 \wedge \neg x_3 = \{000\}$.

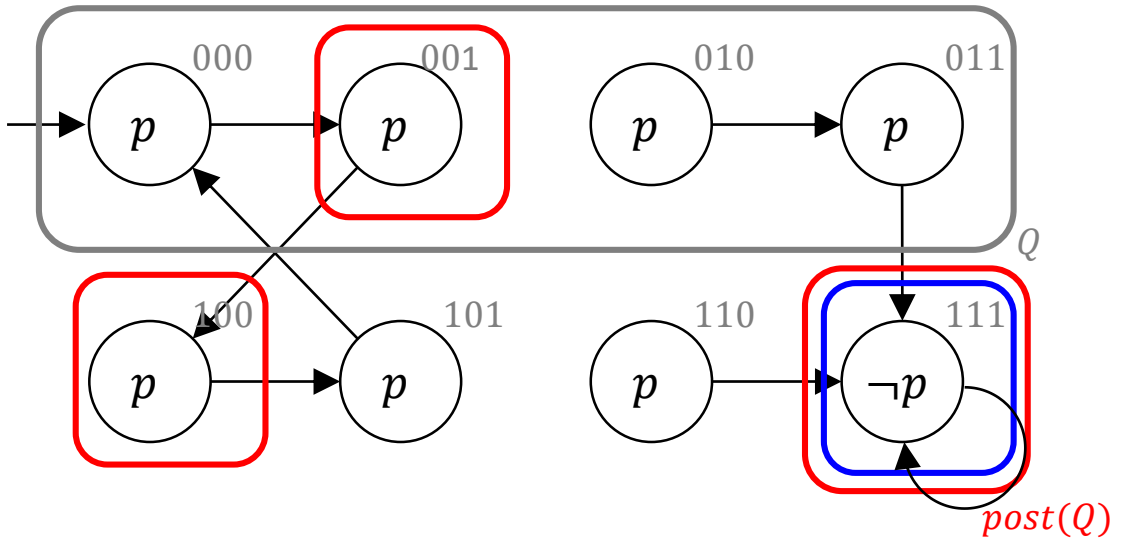
ϕ is UNSAT

Invariant checks first bit: $I = \neg x_1$

if $A \wedge B$ is SAT **then**
  if $Q = S_0$ then return "$M \not\models \mathrm{AG}\, p$";
  increase k
 $Q := S_0(s_0)$;
else
  compute interpolant I for A and B;
  if $I(s_0) \rightarrow Q$ then return "$M \models \mathrm{AG}\, p$";
 $Q := Q \vee I(s_0)$;

$x_1x_2x_3$

Example $\mathbf{AG}\ p$



$\phi = Q(s_0) \wedge R(s_0, s_1) \wedge$
$\wedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \vee_{i=1}^{k} \neg p(s_i)$.

$\boldsymbol{k = 1}$.

$Q = \neg x_1 = \{000,001,010,011\}$.

ϕ is SAT

if $A \wedge B$ is SAT **then**

  if $Q = S_0$ then return "$M \not\models \mathrm{AG}\, p$";

  increase k

 $Q := S_0(s_0)$;

else

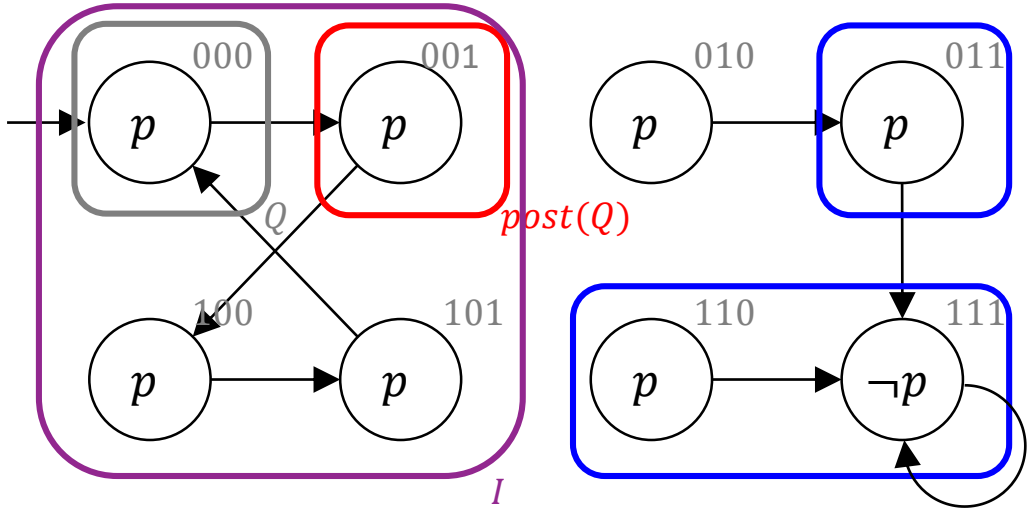
  compute interpolant I for A and B;

  if $I(s_0) \rightarrow Q$ then return "$M \models \mathrm{AG}\, p$";

 $Q := Q \vee I(s_0)$;

$x_1x_2x_3$

Example $\mathbf{AG}\ p$



$\phi = Q(s_0) \wedge R(s_0, s_1) \wedge$
$\wedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \vee_{i=1}^{k} \neg p(s_i)$.

$\boldsymbol{k = 2.}$

$Q = \neg x_1 \wedge \neg x_2 \wedge \neg x_3 = \{000\}.$

ϕ is UNSAT

Invariant checks 2nd bit: $I = \neg x_1$

if $A \wedge B$ is SAT **then**

  if $Q = S_0$ then return "$M \not\models \mathrm{AG}\, p$";

  increase k

 $Q := S_0(s_0)$;

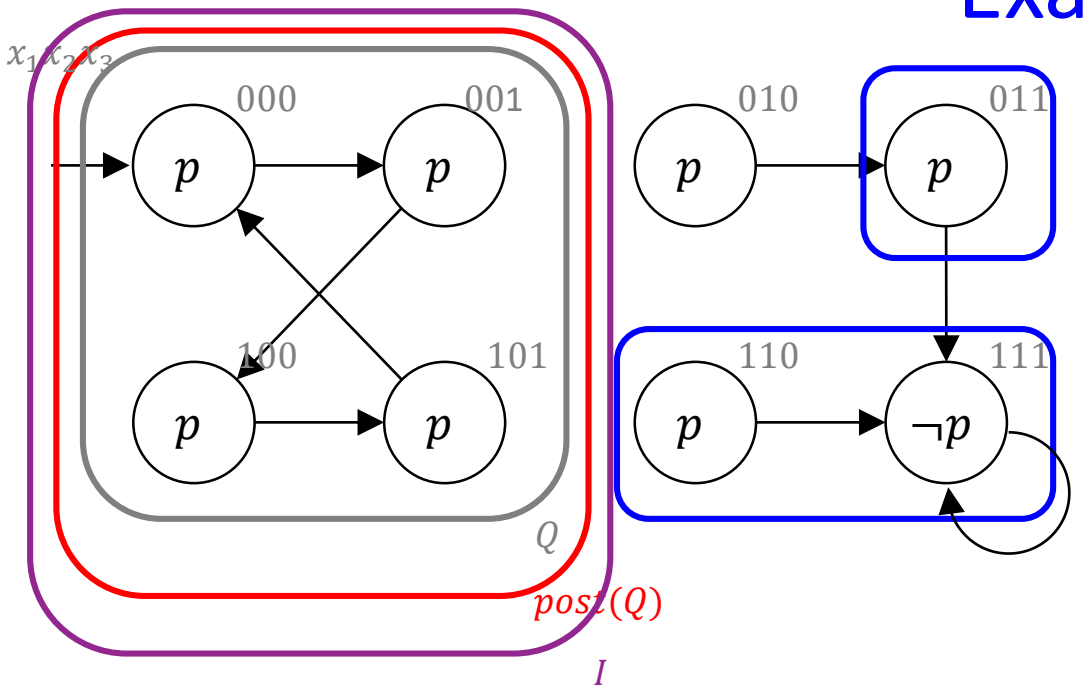
else

  compute interpolant I for A and B;

  if $I(s_0) \rightarrow Q$ then return "$M \models \mathrm{AG}\, p$";

 $Q := Q \vee I(s_0)$;

Example $\mathbf{AG}\ p$



if $A \wedge B$ is SAT **then**

  if $Q = S_0$ then return "$M \not\models \mathrm{AG}\, p$";

  increase k

 $Q := S_0(s_0)$;

else

  compute interpolant I for A and B;

  if $I(s_0) \rightarrow Q$ then return "$M \models \mathrm{AG}\, p$";

 $Q := Q \vee I(s_0)$;

$\phi = Q(s_0) \wedge R(s_0, s_1) \wedge$
$\wedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \vee_{i=1}^{k} \neg p(s_i)$.

$\boldsymbol{k = 2.}$

$Q = \neg x_2 = \{000, 001, 100, 101\}$

ϕ is UNSAT

$\boldsymbol{I = \neg x_2 = Q.}$

Algorithm terminates.

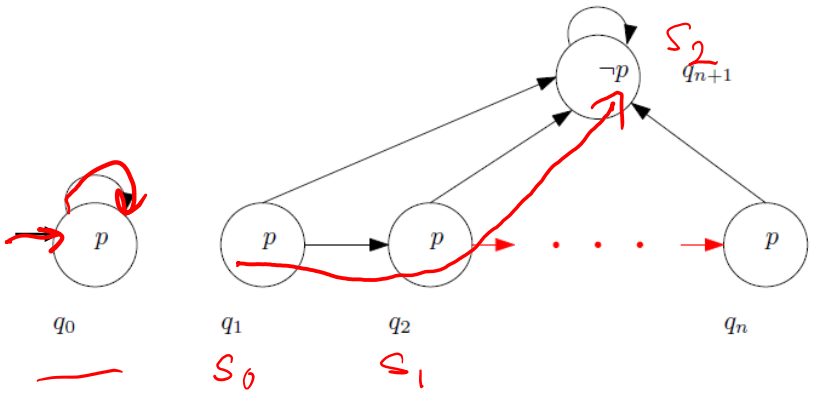
How Did I Pick the Interpolants?

What I did

- Start with $\mathrm{A} = \ postimg(Q)$
- Perform each of the following steps
  1. Can I throw away x_3? (Is $(\exists x_3.A) \cap B = \emptyset$?) If yes, $A \ := \exists x_3.A$
  2. Can I throw away x_2? If yes, $A \ := \exists x_2.A$
  3. Can I throw away x_1? If yes, $A \ := \exists x_1.A$

(This hack only works because the postimg(Q) is a state or a cube!)

Model checking - HW2 - Frequent mistakes



2c

Suppose that q_0 is the initial state. The new formula for the initial states is S'_0.

- What is the smallest k such that k-induction can prove the property correct?
- Suppose $n = 2$. Choose an appropriate k and show the k-induction formula using R, S'_0 and p

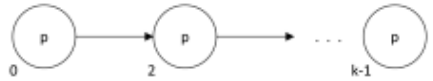
Correct answer : $k = n + 1$ (understanding k as number of states of a path).

A common mistake was answering $k = 2$.

doesn't work!

ψ is satisfiable.

k-induction as Satisfiability



Base. all paths of length *k* from S_0 are labeled *p*

This is BMC! $\phi = S_0(s_0) \wedge \bigwedge_{i=0}^{k-2} R(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k-1} \neg p(s_i)$

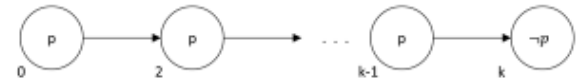
Induction. all paths of length *k* labeled with all *p*'s are followed by a *p*

$$\psi = \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge \bigwedge_{i=0}^{k-1} p(s_i) \wedge \neg p(s_k)$$

Formula satisfiable iff there is a counterexample

$s_0 \rightarrow s_1 \rightarrow s_2$

p p ¬p



HW 3

Property-Directed Reachability
or IC3

Aaron Bradley

PDR

Property-Directed Reachability or IC3

- Makes no copies of transition relation – memory efficient
- Overapproximate postimage (like interpolation)

PDR: Data Structures & Invariants

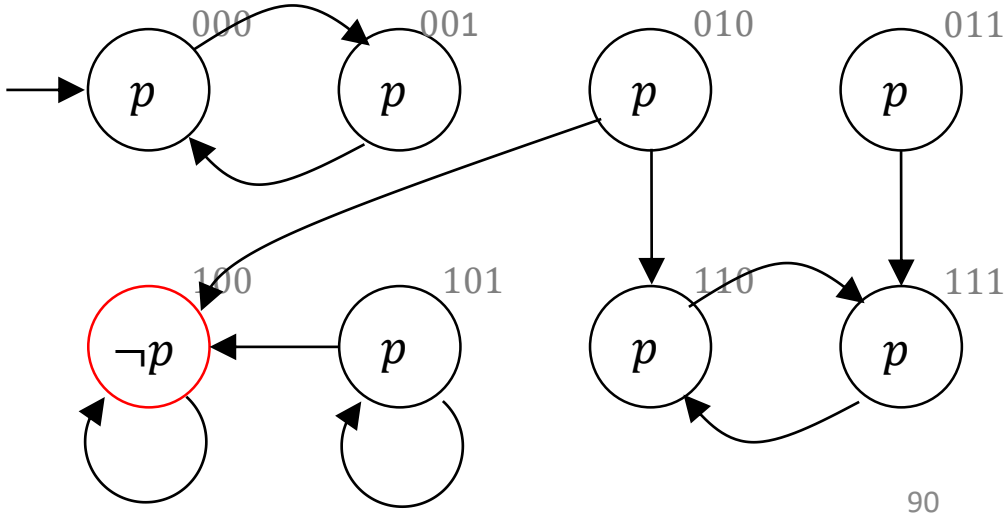
Data Structure

Clause: Disjunction of literals. Cube: Conjunction of literals.

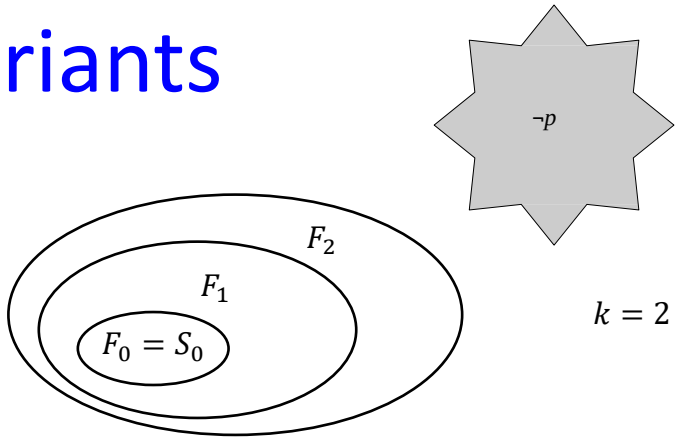
Clause and cubes signify a set of states. Longer clauses – more states. Longer cubes – fewer states

Formulas $F_0, \ldots, F_k$ over V, stored as sets of Clauses (Sets $F_0, \ldots, F_k \subseteq S$)

Example



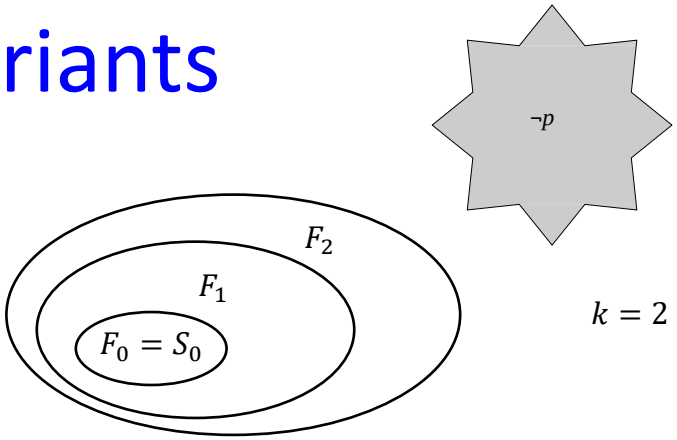
PDR: Data Structures & Invariants



Invariants

- I1: $S_0 \rightarrow F_0$ $(S_0 \subseteq F_0)$
- I2: $F_i \rightarrow F_{i+1}$ $(F_i \subseteq F_{i+1})$
  - I2': $F_i = F_{i+1} \wedge c_{i1} \wedge \cdots \wedge c_{in}$
- I3: $F_i \wedge \neg P = FALSE$ $(F_i \wedge \neg P = \emptyset)$
- I4: $F_i \wedge R \rightarrow F'_{i+1}$ $(postimg(F_i) \subseteq F_{i+1})$

PDR: Data Structures & Invariants



Invariants

- I1: $S_0 = F_0$ $(S_0 = F_0)$
- I2: $F_i \rightarrow F_{i+1}$ $(F_i \subseteq F_{i+1})$
  - I2': $F_i = F_{i+1} \wedge c_{i1} \wedge \cdots \wedge c_{in}$
- I3: $F_i \wedge \neg P = FALSE$ $(F_i \wedge \neg P = \emptyset)$
- I4: $F_i \wedge R \rightarrow F'_{i+1}$ $(postimg(F_i) \subseteq F_{i+1})$

Facts

- $\forall 0 < i \leq k$: There is no trace from F_i to $\neg p$ of $k - i$ edges or less (I3,I4)
- There are no counterexamples of length $\leq k$
- If $F_i = F_{i+1}$ then system is correct. (By I3, I4, F_i is an inductive invariant)

PDR: Notation

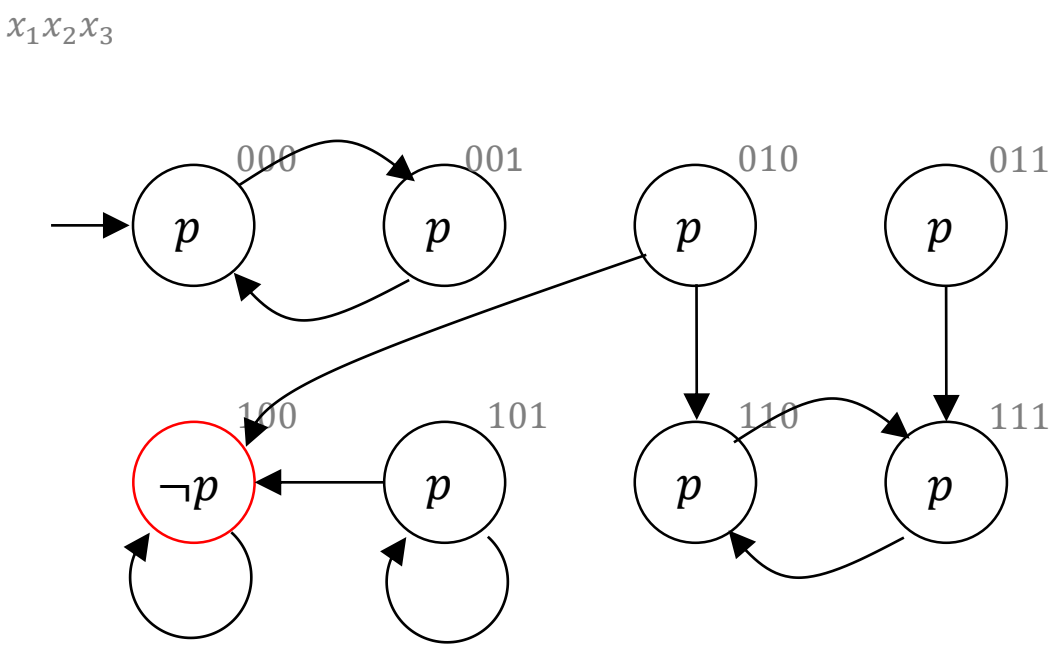
Notation

- s := SAT($F_i \wedge R \wedge \neg P'$):
  - $s :=$ $FALSE$ if ¬SAT($F_i \wedge R \wedge \neg P'$)
  - s := a state satisfying in F_i with an edge to a state in $\neg P$ in otherwise

Definition

- $I \subseteq S$ is inductive if
  1. $S_0 \rightarrow I$ $(S_0 \subseteq I)$
  2. $I \wedge R \rightarrow I'$ $(postimage(I) \subseteq I)$
- $I \subseteq S$ is inductive relative to F if
  1. $S_0 \rightarrow I$ $(S_0 \subseteq I)$
  2. $I \wedge F \wedge R \rightarrow I'$ $(postimage(F \cap I) \subseteq I)$

Relative Inductiveness

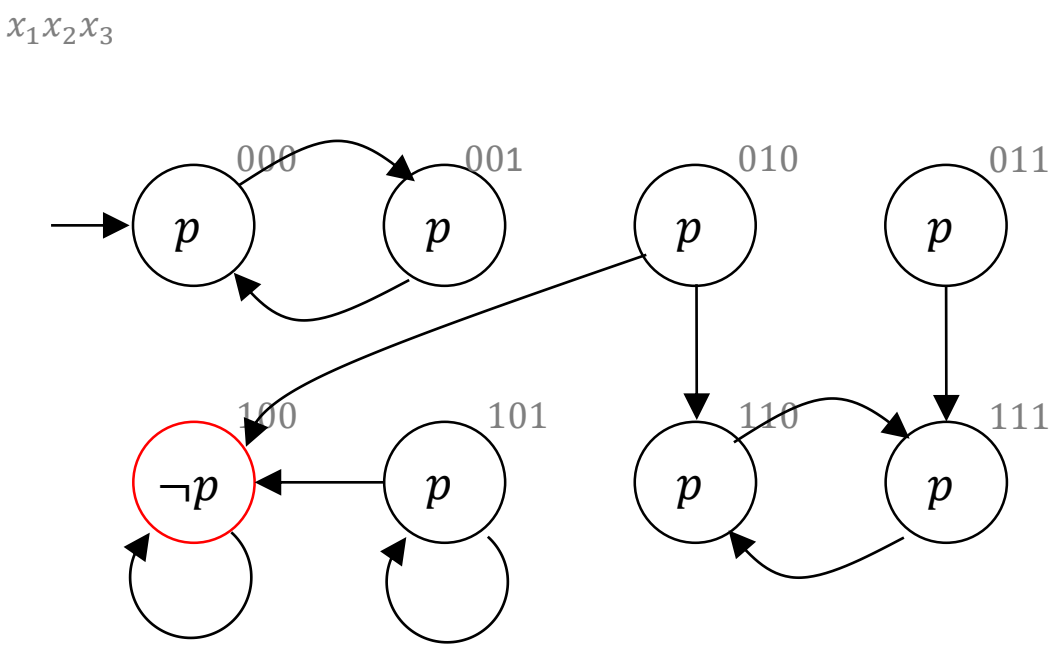


Inductive facts

- Is $\neg x_1$ inductive?

- Is $\neg x_2$ inductive?

Relative Inductiveness



Inductive facts

- Is $\neg x_1$ inductive?
  - $S_0 \rightarrow \neg x_1$
  - $\neg x_1 \wedge R \rightarrow \neg x'_1$ is false
  - No!
- Is $\neg x_2$ inductive?
  - $S_0 \rightarrow \neg x_2$
  - $\neg x_2 \wedge R \rightarrow \neg x'_2$
  - Yes!
- Is $\neg x_1$ inductive relative to x_2?
  - $S_0 \rightarrow \neg x_1$
  - $\neg x_2 \wedge \neg x_1 \wedge R \rightarrow \neg x'_1$
  - **Yes**!

Idea: Find (relatively) inductive facts.

PDR, First Version

function PDR(Model M)

if SAT($S_0 \wedge \neg P$) **or** SAT($S_0 \wedge R \wedge \neg P'$) **then FAIL**

F_0 := S_0; F_1 := P; k :=1;

while(true)

while(s := SAT($F_k \wedge R \wedge \neg P'$))

removeBad(k, s)

k++; F_k := P

if $\exists 0 \leq i < k-1 : F_i = F_{i+1}$ **then SUCCEED**

remove states in F_k with edge to $\neg P$

// post: $\neg SAT(F_i \wedge s)$

function removeBad($i \in N$, state s)

if SAT($S_0 \wedge s$) **then FAIL**

while(t := SAT($F_{i-1} \wedge R \wedge s'$))

removeBad($i-1, t$)

$\forall 0 < j \leq i : F_j := F_j \wedge \neg s$

remove states in F_i with path to $\neg P$ of length $k - i + 1$

PDR, First Version

function PDR(Model M)

  if SAT($S_0 \wedge \neg P$) **or** SAT($S_0 \wedge R \wedge \neg P'$) **then FAIL**

  $F_0 := S_0$; $F_1 := P$; $k :=1$;

  while(true)

    while($s :=$ SAT($F_k \wedge R \wedge \neg P'$))

      removeBad(k, s)

    k++; $F_k := P$

    if $\exists 0 \leq i < k-1 : F_i := F_{i+1}$ **then SUCCEED**

// post: $\neg SAT(F_i \wedge s)$

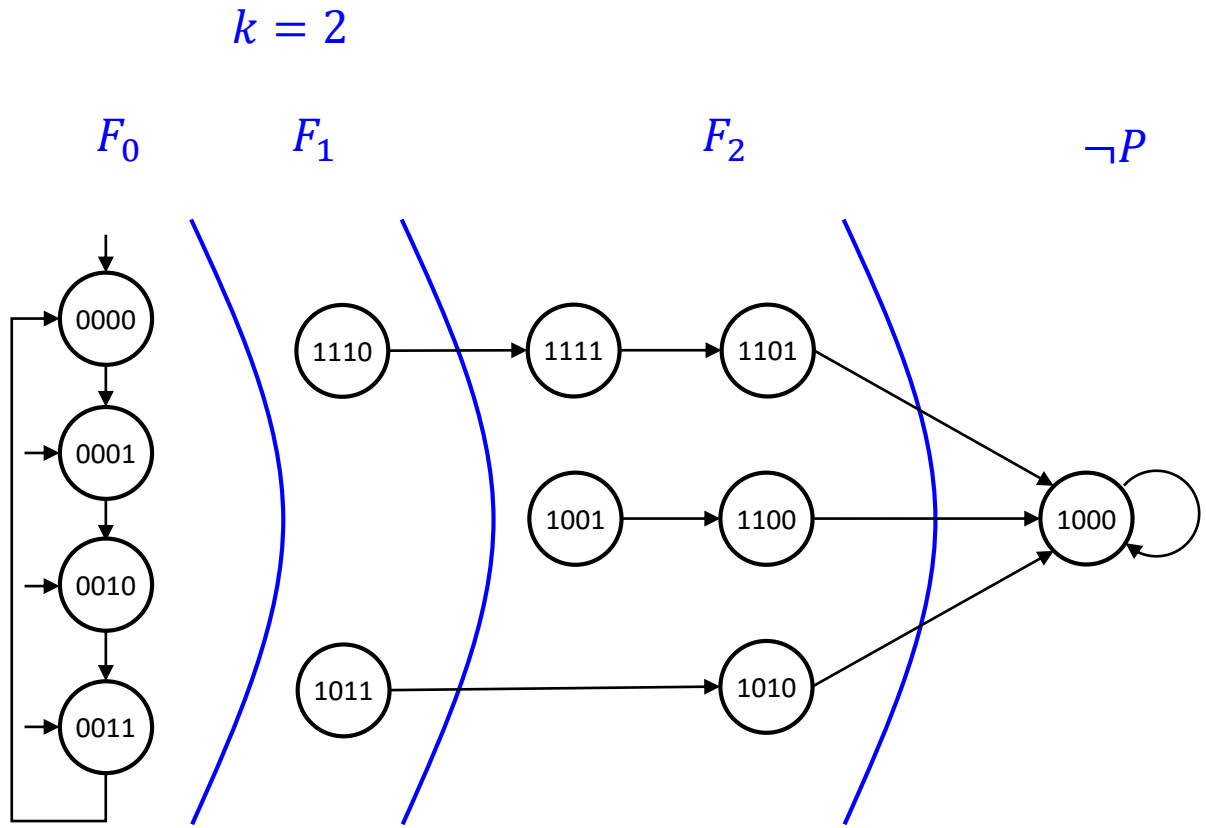
function removeBad($i \in N$, state s)

  if SAT($S_0 \wedge c$) **then FAIL**

  while($t :=$ SAT($F_{i-1} \wedge R \wedge s'$))

    removeBad($i-1, t$)

  $\forall 0 < j \leq i : F_j := F_j \wedge \neg s$



PDR, First Version

function PDR(Model M)

  if SAT($S_0 \wedge \neg P$) **or** SAT($S_0 \wedge R \wedge \neg P'$) **then FAIL**

  F_0 := S_0; F_1 := P; k :=1;

  while(true)

    while(s := SAT($F_k \wedge R \wedge \neg P'$))

      removeBad(k, s)

    k++; F_k := P

    if $\exists 0 \leq i < k-1 : F_i := F_{i+1}$ **then SUCCEED**

// post: $\neg SAT(F_i \wedge s)$

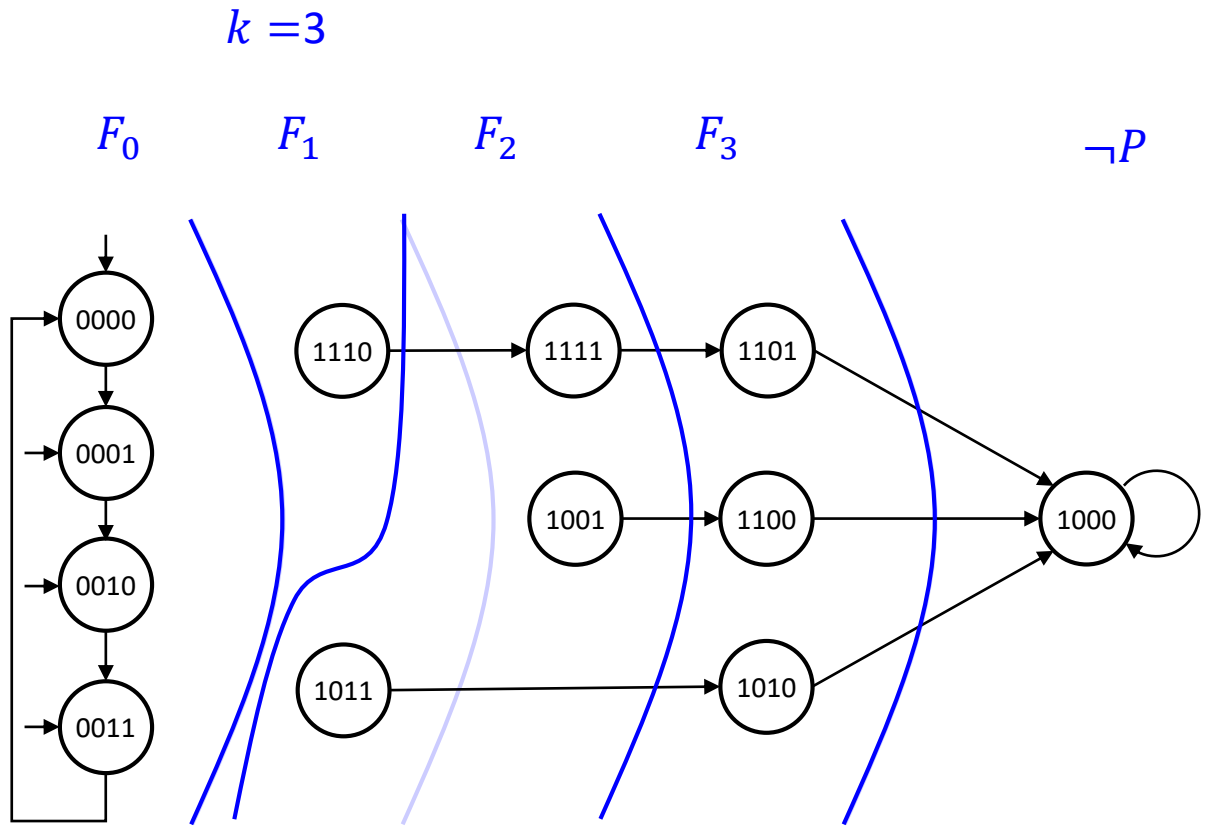
function removeBad($i \in N$, state s)

  if SAT($S_0 \wedge c$) **then FAIL**

  while(t := SAT($F_{i-1} \wedge R \wedge s'$))

    removeBad($i-1, t$)

  $\forall 0 < j \leq i : F_j := F_j \wedge \neg s$



PDR, First Version

function PDR(Model M)

  if SAT($S_0 \wedge \neg P$) **or** SAT($S_0 \wedge R \wedge \neg P'$) **then FAIL**

  F_0 := S_0; F_1 := P; k :=1;

  while(true)

    while(s := SAT($F_k \wedge R \wedge \neg P'$))

      removeBad(k, s)

    k++; F_k := P

    if $\exists 0 \leq i < k-1 : F_i := F_{i+1}$ **then SUCCEED**

// post: $\neg SAT(F_i \wedge s)$

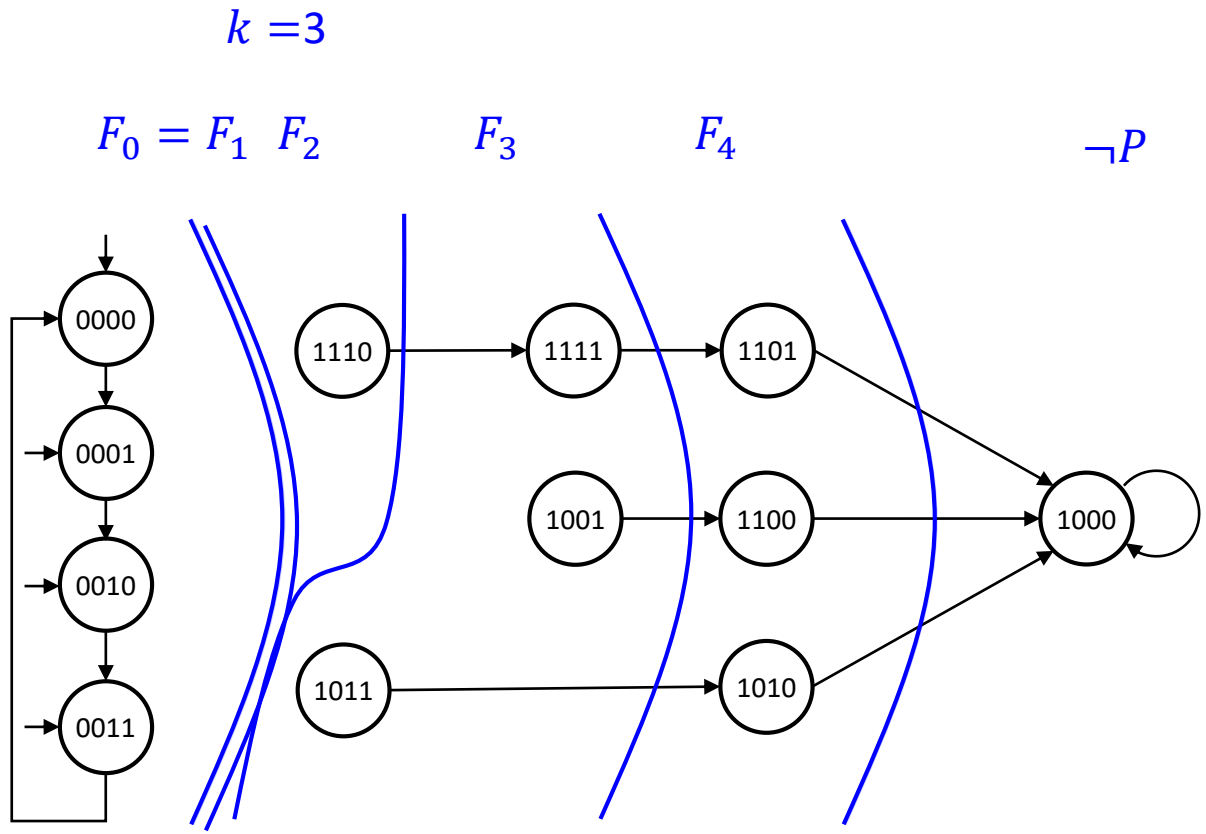
function removeBad($i \in N$, state s)

  if SAT($S_0 \wedge c$) **then FAIL**

  while(t := SAT($F_{i-1} \wedge R \wedge s'$))

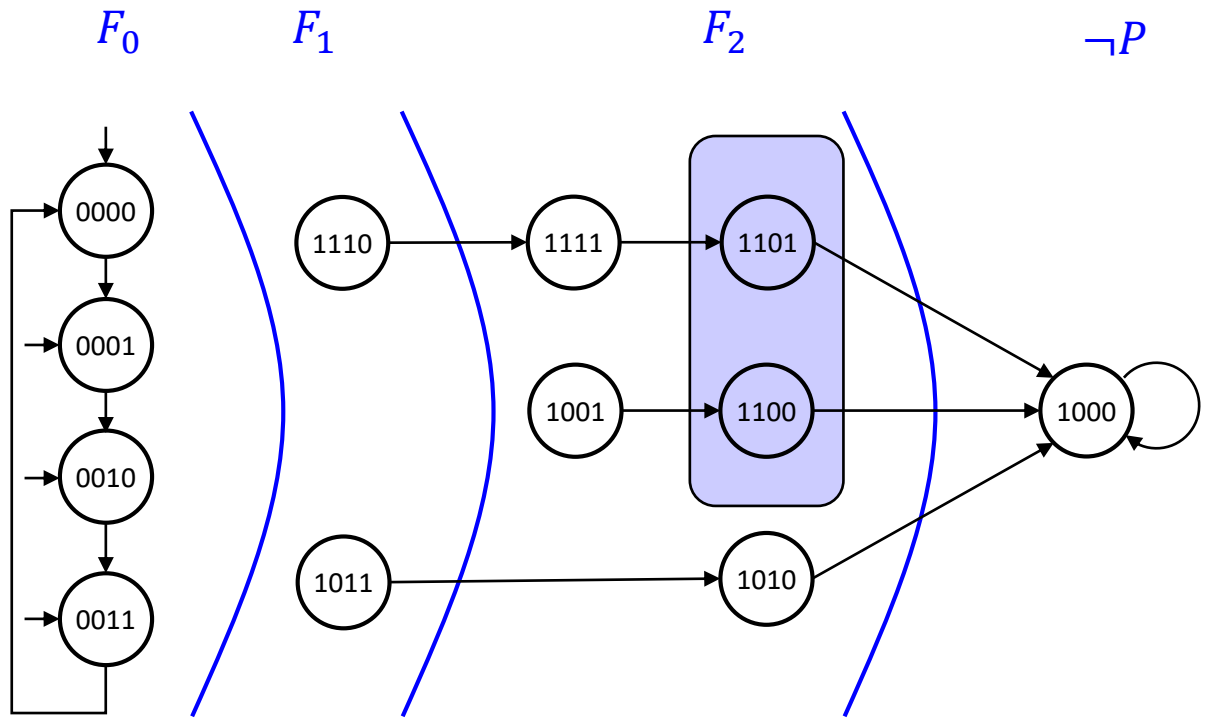
    removeBad($i-1, t$)

  $\forall 0 < j \leq i : F_j := F_j \wedge \neg s$



Drawback $k = 2$

- The first version considers every state individually
- But similar states behave similarly. Example: 1100 and 1101. Generalize to 110- = $x_1 \wedge x_2 \wedge \neg x_3$!



Conditions

- UNSAT($F_1 \wedge R \wedge x'_1 \wedge x'_2 \wedge \neg x'_3$)
- UNSAT($S_0 \wedge x_1 \wedge x_2 \wedge \neg x_3$)

PDR: Naive Generalization

function PDR(Model M)

if SAT($S_0 \wedge \neg P$)) **or** $SAT(S\_0 \wedge R \wedge \neg P')$ **then FAIL**

F_0 := S_0; F_1 := P; k :=1;

while(true)

while(s := SAT($F_k \wedge R \wedge \neg P'$))

removeBad(k, s)

k++; F_k := P

if $\exists 0 \leq i < k-1 : F_i := F_{i+1}$ **then SUCCEED**

// post: $\neg SAT(F_i \wedge s)$

function removeBad($i \in N$, state s)

if SAT($S_0 \wedge c$) **then FAIL**

while(t := SAT($F_{i-1} \wedge R \wedge s'$))

removeBad($i-1, t$)

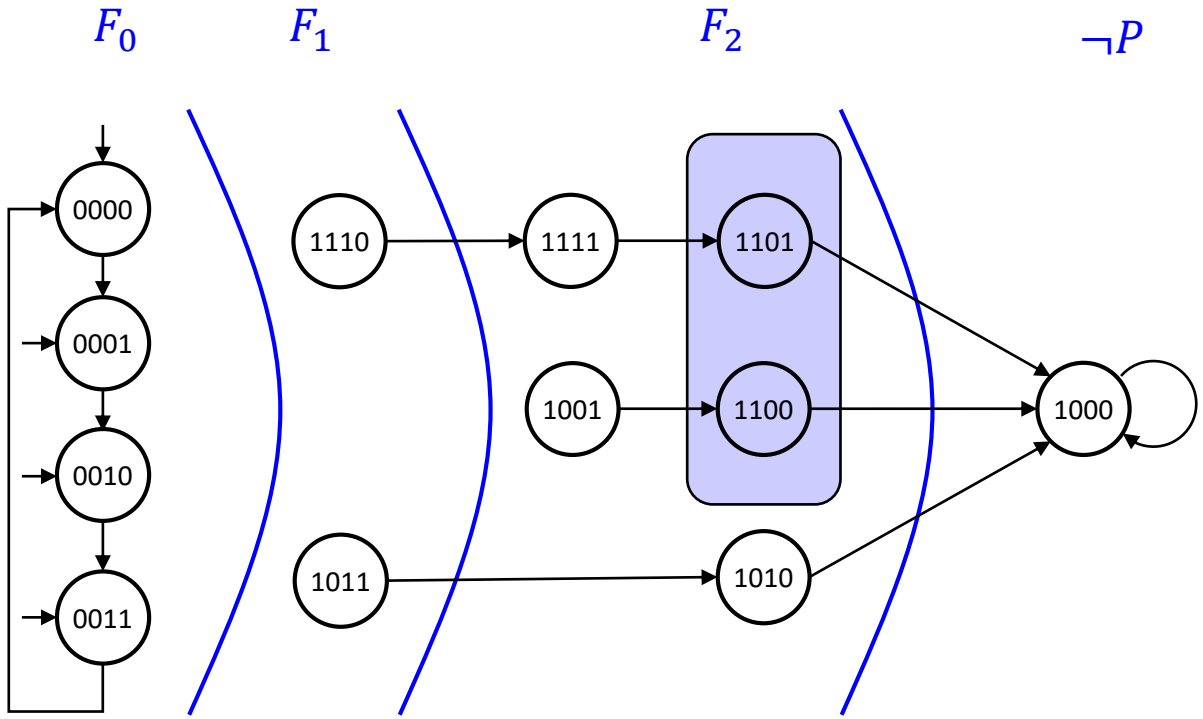
g := generalizeNaive(i, s)

$\forall 0 < j \leq i : F_j := F_j \wedge \neg g$

function generalizeNaive(i, state s)

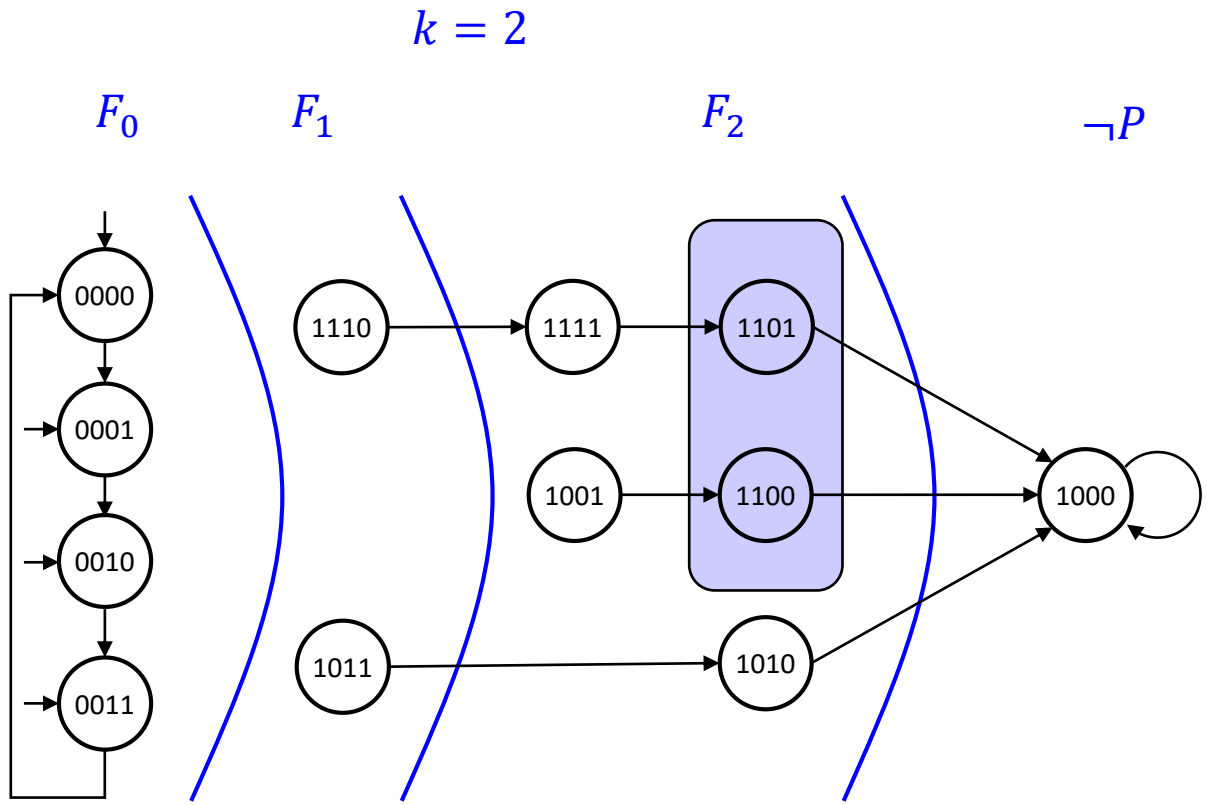
return a shortest cube c such that

- $c \leftarrow s$
- $\neg \mathrm{SAT}(\mathrm{F_{i-1}} \wedge R \wedge c')$
- $\neg \mathrm{SAT}(S_0 \wedge c)$



Generalize Further?

- Generalize to 110-
- For $f = x_1 \wedge x_2$, we have SAT($F_1 \wedge R \wedge f$)
- Same for $\mathrm{f} = \mathrm{x}_2 \wedge \neg x_3$, $\mathrm{f} = x_2 \wedge \neg x_3$
- State 1100 is the problem

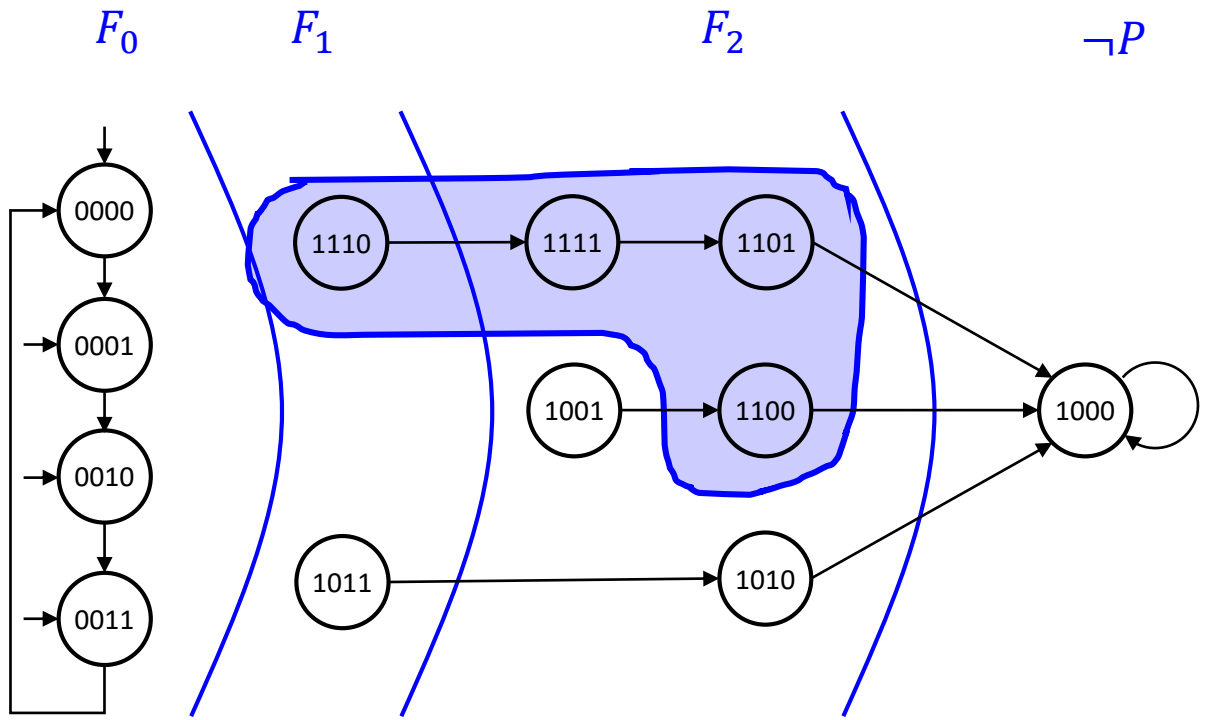


Relative Inductiveness

shortest cube c such that

- $c \leftarrow s$
- $\neg\mathrm{SAT}(\neg c \wedge \mathrm{F}_1 \wedge R \wedge c')$
- $\neg\mathrm{SAT}(S_0 \wedge c)$

$\neg c$ is relative inductive wrt F_1



PDR: Relative Inductiveness

function PDR(Model M)

if SAT($S_0 \wedge \neg P$) **or** $SAT(S_0 \wedge R \wedge \neg P')$ **then FAIL**

F_0 := S_0; F_1 := P; k :=1;

while(true)

while(s := SAT($F_k \wedge R \wedge \neg P'$))

removeBad(k, s)

k++; F_k := P

if $\exists 0 \leq i < k-1 : F_i = F_{i+1}$ **then SUCCEED**

// post: $\neg SAT(F_i \wedge s)$

function removeBad($i \in N$, state s)

if SAT($S_0 \wedge c$) **then FAIL**

while(t := SAT($F_{i-1} \wedge R \wedge s'$))

removeBad($i-1, t$)

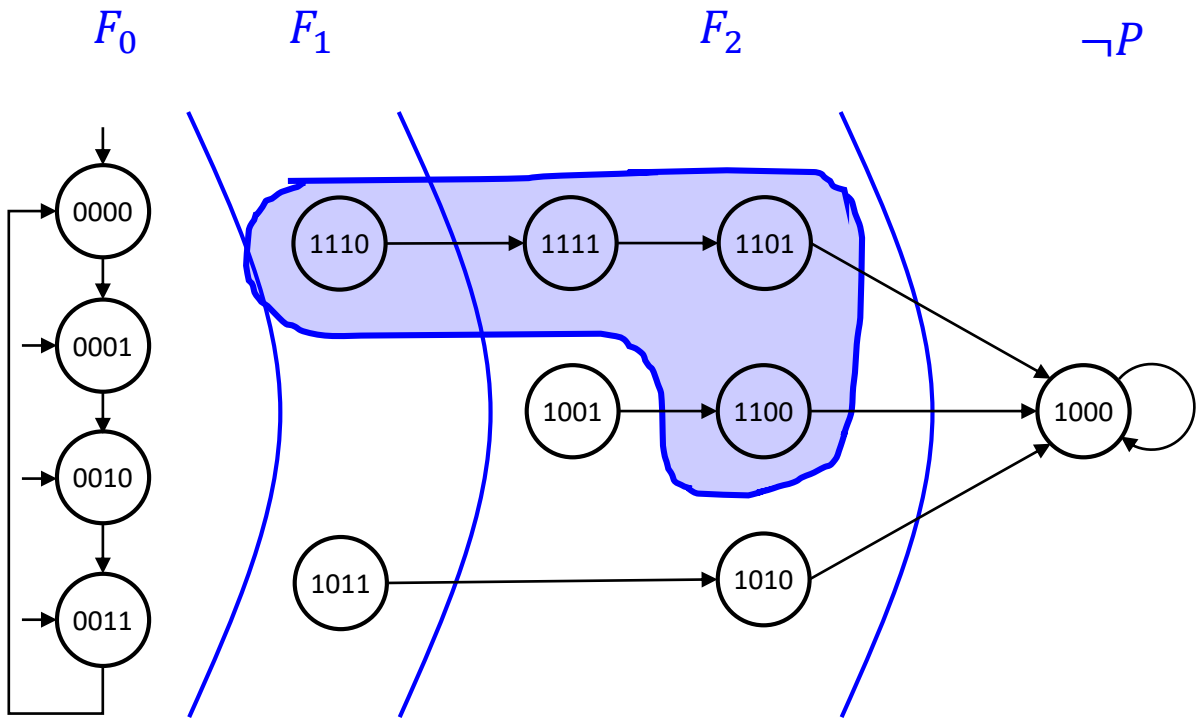
g := generalize(i,s)

$\forall 0 < j \leq i : F_j := F_j \wedge \neg g$

function generalize(i, state s)

return a shortest cube c such that

- $c \leftarrow s$
- $\neg \mathrm{SAT}(\neg c \wedge \mathrm{F_{i-1}} \wedge R \wedge c')$
- $\neg \mathrm{SAT}(S_0 \wedge c)$



Generalization

function generalize(i, state s)

$c := s$

while c changes

let $l_1, \ldots, l_n$ be the literals of c

for $i := 1$ **to** n

$c' = c$ with l_i removed

if relInd(c') **then** $c = c'$

return c

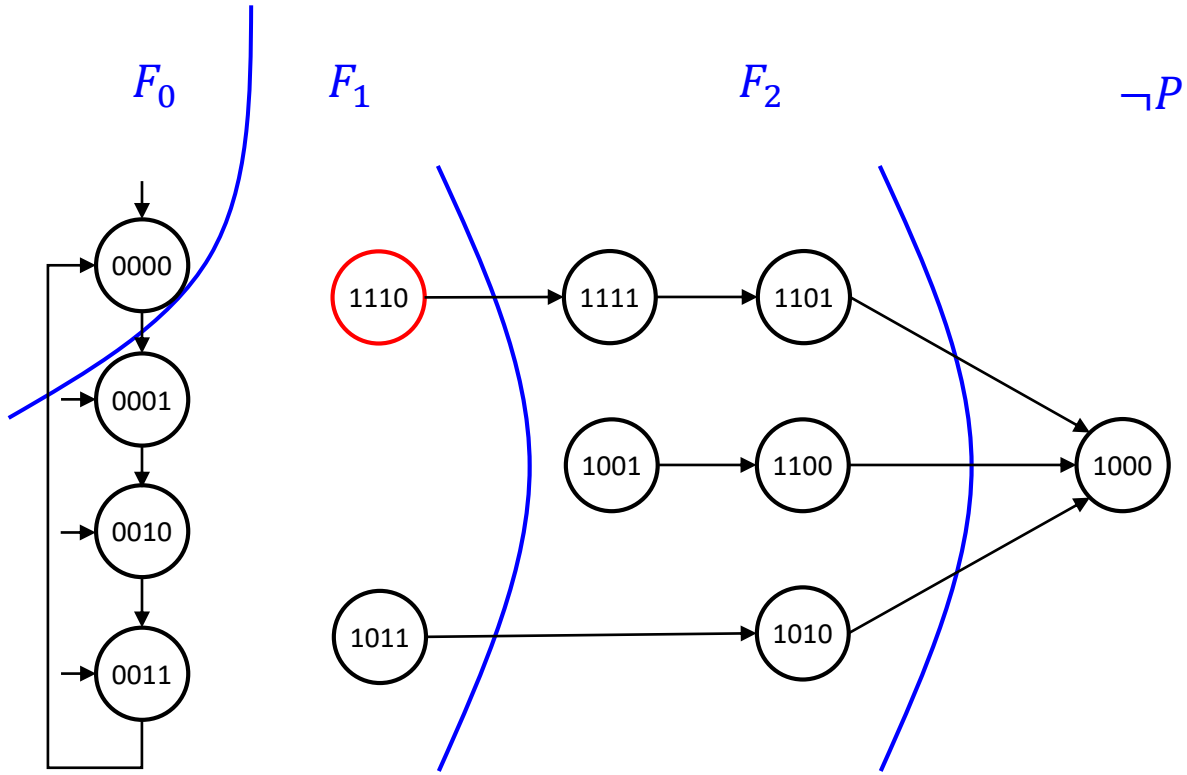
function relInd(cube c)

return $\neg\mathrm{SAT}(\neg c \wedge \mathrm{F_{i-1}} \wedge R \wedge c) \wedge \neg\mathrm{SAT}(S_0 \wedge c)$

Propagate Clauses

Suppose you are removing 1110.
You can generalize to 1---

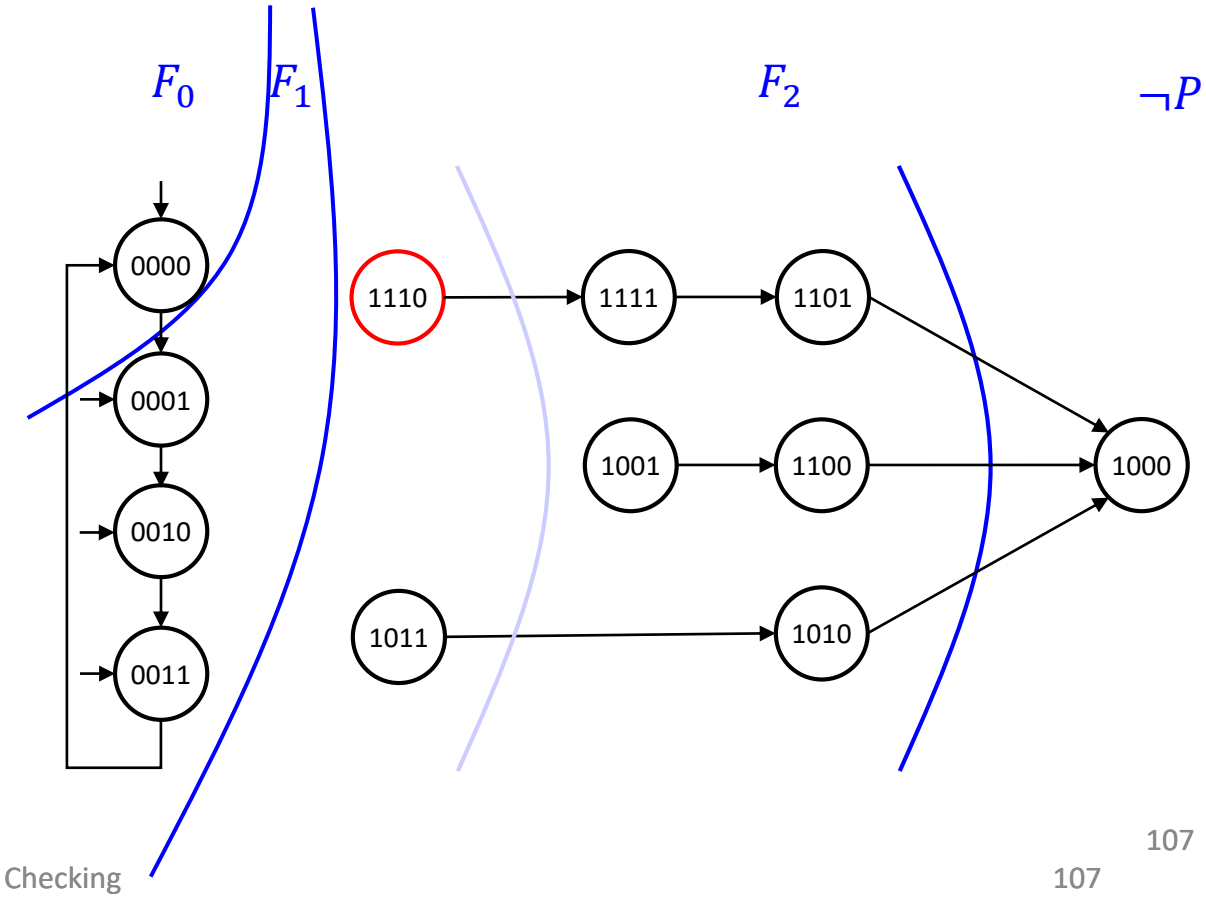
$$F_1 := F_1 \wedge \neg x_1$$



Propagate Clauses

$F_2 \wedge x_1 \notin postimg(F_1)$, so we can add $\neg x_1$ to F_2

UNSAT($F_1 \wedge R \wedge F'_2 \wedge x'_1$)



PDR, Final: Propagate Clauses

function PDR(Model M)
  if SAT($S_0 \wedge \neg P$) **or** $SAT(S_0 \wedge R \wedge \neg P')$ **then FAIL**
  F_0 := S_0; F_1 := P; k :=1;
  while(true)
    while(s := SAT($F_k \wedge R \wedge \neg P'$))
      removeBad(k, s)
    k++; F_k := P
    propagateClauses(k)
    if $\exists 0 \leq i < k-1 : F_i = F_{i+1}$ **then SUCCEED**

// post: $\neg SAT(F_i \wedge s)$
function removeBad($i \in N$, state s)
  if SAT($S_0 \wedge s$) **then FAIL**
  while(t := SAT($F_{i-1} \wedge R \wedge s'$))
    removeBad($i-1, t$)
  g := generalize(i,s)
  $\forall 0 < j \leq i : F_j := F_j \wedge \neg g$

function generalize(i, state s)
  return a shortest cube c such that
    - $c \leftarrow s$
    - $\neg c$ inductive relative to F_{i-1}

function propagateClauses(k)
  for i := 1 **to** $k-1$
    for every clause $c \in F_i$
      if $\neg \mathrm{SAT}(F_i \wedge R \wedge \neg c')$
        $F_{i+1} := F_{i+1} \wedge c$

Further Ideas

- Equivalence of frames = syntactic check
  - Use implication and subsumption to simplify clauses
  - Check Mischchenko paper to see if we add clauses when they are subsumed
- Use a queue to find long counterexamples quickly

Performance

Hardware Model Checking Competition 2020

1. AVR 11 variants of IC3+[abstraction], 2x BMC, 3x k-induction
2. AVY interpolation + PDR
3. nuXmv “portfolio”, including IC3
4. Pono: protfolio, including BMC, k-induction, interpolation, IC3

Literature

Literature

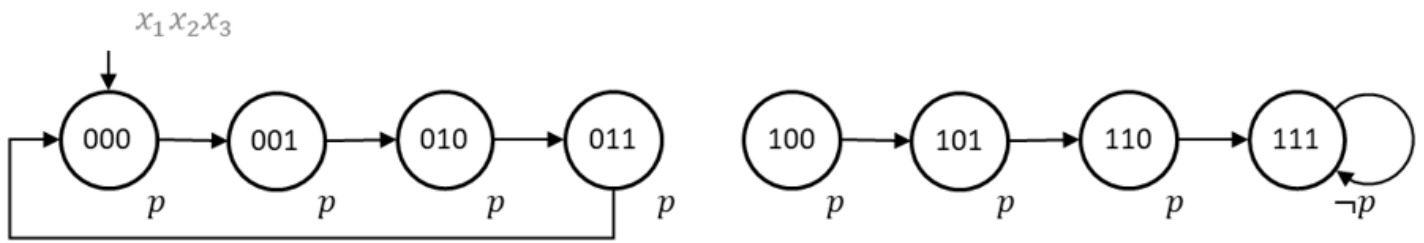
- A. R. Bradley, SAT-Bassed Model Checking without Unrolling, VMCAI 2011. http://ecee.colorado.edu/~bradleya/ic3/ic3_bradley.pdf
- N. Een, A. Mishchenko, R. Brayton, Efficient Implementation of Property Directed Reachability, FMCAD 2011. https://people.eecs.berkeley.edu/~alanmi/publications/2011/fmcad11_pdr.pdf
- F. Somenzi, Aaron R. Bradley: IC3: where monolithic and incremental meet. FMCAD 2011: 3-8. http://theory.stanford.edu/~arbrad/papers/ic3_tut.pdf
- A. R. Bradley: Understanding IC3. SAT 2012: 1-14. https://theory.stanford.edu/~arbrad/papers/Understanding_IC3.pdf

Model Checking Homework 4

Deadline: 22 April 4:00pm
Send solution to: modelchecking@iaik.tugraz.at

Question hour
Mon 19 Apr 10 AM, discord

Consider the following Kripke structure K.



We will use the variant of PDR shown in class to prove that the property is true. (If you want to use a different variant, talk to us first.) Clearly indicate the steps. Indicate at least the frames at every step, the function calls, and the generalizations you use. The two subtasks differ in how to build the generalizations. We will use a generalization algorithm that finds a relative inductive clause from a fixed order of preference.

Task 3a. [5 points]. Suppose the state is $s = l_1 \wedge l_2 \wedge l_3$, where $l_i = x_i$ or $l_i = \neg x_i$. The following is a list of all cubes that are implied by s.

$$true, l_1, l_2, l_1 \wedge l_2, l_3, l_1 \wedge l_3, l_2 \wedge l_3, l_1 \wedge l_2 \wedge l_3$$

The generalization algorithm takes the first cube from this list such that its negation is relative inductive with respect to F_i. This order should allow you to find the inductive invariant quickly. (Note that false is not relative inductive to any clause, because of initialization.)

Task 3a. [5 points]. Now use the following order, which should lead to more iterations:

$$true, l_3, l_2, l_3 \wedge l_2, l_1, l_3 \wedge l_1, l_2 \wedge l_1, l_1 \wedge l_2 \wedge l_3.$$